JACOBS
UNIVERSITY

Towards Flexiformal Mathematics

by

Mihnea Iancu

A thesis submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy
in Computer Science

Approved Dissertation Committee

Prof. Dr. Michael Kohlhase

Jacobs University Bremen, Germany

Prof. Dr. Herbert Jaeger

Jacobs University Bremen, Germany

Prof. Dr. William M. Farmer

McMaster University, Canada

Priv.-Doz. Dr. Florian Rabe

Jacobs University Bremen, Germany

Date of defense: Jan 5, 2017

Computer Science & Electrical Engineering

Statutory Declaration

(Declaration on Authorship of a Dissertation)

I, Mihnea Iancu, hereby declare, under penalty of perjury, that I am aware of the consequences of a deliberately or negligently wrongly submitted affidavit, in particular the punitive provisions of §156 and §161 of the Criminal Code (up to 1 year imprisonment or a fine at delivering a negligent or 3 years or a fine at a knowingly false affidavit).

Furthermore I declare that I have written this PhD thesis independently, unless where clearly stated otherwise. I have used only the sources, the data and the support that I have clearly mentioned.

This PhD thesis has not been submitted for the conferral of a degree elsewhere.

Place Date

Signature

Parts of this thesis are based on or closely related to previously published material or material that is prepared for publication at the time of this writing. These are [IKR], [IK15b], [IK15a], [KI14], [IKP14], [IJKW14], [GIJ+16], [IKRU13] and [IKR11]. Parts of this work are collaborations with other researchers: For Part I with Michael Kohlhase, for Part II with Michael Kohlhase and Florian Rabe and for Part III with Constantin Jucovschi, Michael Kohlhase, Enxhell Luzhnica, Akbar Oripov, Corneliu Prodescu, Florian Rabe and Tom Wiesing. In each case the precise connection to this work is detailed in the relevant passages of the text.

Abstract

The application of computer-based methods to mathematics, while meaningful, is constrained by the fact the most mathematical knowledge exists in forms that can only be understood by humans. In order for it to be also understood by machines, those aspects of mathematical knowledge that are relevant for machine-driven applications need to be made explicit.

One important aspect of mathematics is the underlying semantics which can be made understandable to machines by formalizing it. Formalization makes explicit the implicit definitions and inference steps that occur naturally in mathematical documents. But, despite numerous attempts, only a small fragment of mathematics has been formalized because formalization is prohibitively expensive. Furthermore, existing formalized mathematics typically lacks in other aspects such as presentation information and narrative structure that are common in mathematics and critical for human-oriented practical applications. Therefore, existing applications for formal mathematics are mostly limited to verification so there is little practical incentive for formalization.

Relying on the idea of flexiformality we propose *flexiformalizing* mathematics which addresses the bottlenecks discussed above in two ways. First, flexiformality allows content of flexible formality, so it minimizes the starting cost of flexiformalization compared to formalization. Second, flexiformality involves co-representing the narration, structure and meaning of mathematical knowledge. Therefore, it forms a basis for not only machine processing but also for building practical, human-oriented applications. We call the result of flexiformalization as described here *flexiformal knowledge* and we believe mathematical knowledge is fundamentally flexiformal.

In this thesis, we survey both informal and formal mathematics to identify flexiformal phenomena in mathematical practice. Then, we design and implement a formal language for flexiformal knowledge to adequately represent them. Furthermore, we evaluate the language and implementation by importing existing mathematical libraries at different levels of formality and developing applications for them. Finally, we integrate the libraries and applications into a generic system as an extensive case study in flexiformal knowledge management.

First and foremost, I would like to express my gratitude to Michael Kohlhase for his continuous guidance, support and encouragement, as well as for suggesting this research topic in the first place. I am especially indebted to Florian Rabe whose comments and advice have always proved compelling and insightful. I am also grateful to Herbert Jaeger and William Farmer for their support and feedback.

Finally, I would like to thank the members of the KWARC group whose ideas and remarks proved often fruitful and always interesting.

Contents

Chapter 1

Introduction

Throughout history, knowledge has been a crucial building block for human civilization, and its creation, distribution and consumption have been driving forces for human progress. Mathematics, in particular, is among the oldest areas of human knowledge and is the foundation for most of modern science. Moreover, mathematical knowledge still grows relentlessly as the total amount of published mathematics has been estimated to double every ten to fifteen years [Odl95].

The digital revolution has already transformed mathematical practice. Most mathematical knowledge is rendered and read using computer systems such as PDF readers and web browsers. It is organized in electronic repositories of mathematical knowledge such as arXiv [ArX], euDML [EuD], Math Reviews [MRv] or ZentralBlatt math [ZBM]. Furthermore, computer algebra systems (CAS) such as Mathematica [Wol02], Maple [Mpl16], SageMath [Sag16], GAP [GAP16] or MuPAD [MPD16] are routinely used for computation.

However, computer support for mathematics is limited by the fact that most mathematical knowledge exists as human-oriented content that is inaccessible to computer processing. Currently, most mathematics is laid down in the form of documents ranging from research articles to engineering whitepapers to math textbooks. These are often encoded electronically as digitizations of printed material or born-digital in the form of PDF generated from LaTeX (predominatly in math research) but also other formats (in education and engineering). However, they are usually targeted at and exclusively consumed by human readers while their semantics is inaccessible to computers systems. In order to support "doing mathematics" by computer we have to recover the knowledge in these documents in a way that computers can act on. [Far08] defines this as *practical expressivity* as opposed to theoretical expressivity which disregards how readily the knowledge can be encoded and used by computer applications. In this thesis, motivated by building practical applications, we will focus on practical expressivity. We identify two distinct, existing approaches towards achieving that goal.

The first is to use computer methods to semanticize the existing mathematical documents. This process is essentially the syntactic and semantic analysis phase of natural language processing applied to mathematical documents. The analysis process is traditionally viewed as a transformation from utterances or documents into a knowledge representation language. First-order logic

can seem a plausible target format for mathematical knowledge because it is (together with axiomatic set theory [Ber91]) the generally accepted theoretical foundation of mathematics. But, in practice, phenomena from natural language require more expressive languages.

For example, Montague [Mon74] uses a higher-order logic based on λ-calculus to have a compositional treatment of noun phrases and determiners. Similarly, dynamic logics like DRT [KR93] or DPL [GS91] are used to address structural issues like the accessibility of discourse referents. The examples above concentrate on "general natural language", which in practice means almost exclusively non-STEM documents. While STEM documents and mathematical ones in particular retain most of the linguistic complexities [Bau99, Wol13] of natural language, they have their very own peculiarities that mean traditional natural language processing tools are not directly applicable (see [Wol12]). On the other hand, mathematical documents are typically more rigorous and have more explicit representations of the underlying semantics. Therefore it makes sense to develop specialized representation formats that take these into account.

The second approach is to design computer-processable languages together with associated systems and then to re-develop (part of) mathematics from the ground up in those languages. This has lead to a wide range of formal systems such as Coq [Coq96], Mizar [Miz], Isabelle [Pau94] or HOL Light [Har11]. Each has developed associated libraries of content in that language which generally range in the area of thousands of documents. However, this size is only a fraction of what is routinely produced in the informal world. In terms of services, the main utility of such systems has been mechanically checking proofs to ensure their correctness. Major results include the Flyspeck project [HAB+15] formalizing the proof of the Kepler conjecture in HOL Light [Har11] and Isabelle [Pau05] and the formalization of the Feit-Thompson theorem [GAA+13] in Coq [BC04, Coq96]. A major limitation of such formal systems is that the cost of formalization is very high. Moreover, they require a commitment to one logical language and system with interoperability between systems and libraries being very limited. The QED project, whose goals were outlined in 1994 in the QED manifesto [QED96, QED95], proposed creating a database of all mathematical knowledge, completely formalized and with all proofs formally checked but it never gained traction. In [Wie07b] Freek Wiedijk identifies two major reasons for this failure, the small size of the formalized mathematics community and the lack of a "killer application" application for formalized content.

In this thesis we tackle the problem of mechanizing mathematics and focus, not on the formalization or translation process, but on the underlying representation language or target format. Moreover, we use an application driven approach where the design and implementation of the representation language is informed by the practicality of developing services for it. For restricted domains, language, system and applications are often designed together for a specific goal (e.g. in a proof assistant), but this makes it difficult to re-purpose the system for new applications. This is partly because, often, the aspects of mathematical knowledge that are relevant for mechanization differ between applications. For instance, *i)* computer algebra systems make use of and focus on the computational aspect, *ii)* proof checking requires explicit representation of the semantics of types and definitions but can ignore narrative structures such as sections or paragraphs, *iii)* formula search uses the presentational structure of expressions to match results and

iv) change management can benefit from introspecting statement and document level structural constructs to compute dependencies between them.

In practice, trained mathematicians integrate understanding about all aspects of mathematical knowledge into a mental model. Then, they can use this mental model to perform reasoning and knowledge operations that combine the various aspects in order to understand newly acquired knowledge as well as induce new results. We will call the ability to do this *mathematical literacy* since it is a prerequisite for doing mathematics effectively. Correspondingly, we consider mathematical literacy of an application as its ability to combine aspects of mathematical knowledge, such as abstracting from syntactic differences in its input, then performing inferences, computations or interpretations via non-trivial mappings internally, and finally adapting the result to users background knowledge or notational preferences. Therefore, in order to scale to a multitude of services and allow building math literate applications a representation language for mathematics must be able to capture, at the same time, all aspects of mathematical knowledge that are relevant for practical, machine-driven applications.

Contributing to designing such a language for representing mathematics, in this thesis, we start with and focus on the semantic and narrative aspects. Most notably, we largely ignore the computational aspect and consider it outside the scope of this work. However, related developments concerning simultaneously representing the semantic and computational aspects such as [Far07], [KMR13] and [DIK+16] suggest promising avenues for future work.

Here, we start from the flexiformal ideas introduced in [KK11, Koh13] and build upon the existing representation languages MMT [RK13a] and OMDOC [Koh06]. The flexiformalist view of knowledge representation extends the usual dichotomy of formal versus informal mathematics. It emphasizes the importance of both representations and integration between them. Therefore, a *flexiformal* document merges the human-oriented presentation, which we call the *narrative aspect* with its underlying semantics which we call *content aspect*.

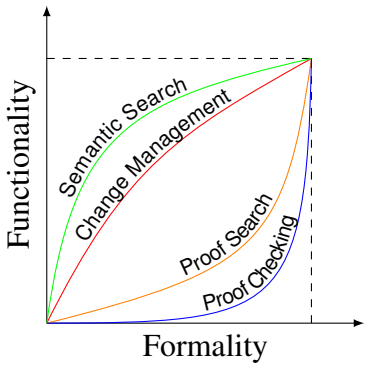
In concrete documents, either (or both) the content and the narrative aspect may be partial. From that perspective, the two common cases occurring in practice, which are typically described as formal and, respectively, informal documents represent two opposite extremes in the flexiformal space. In this thesis we will use the terms *transparent* and *opaque* to describe the explicit and, respectively, lack thereof a representation of either the content or narrative aspect. Therefore, informality of documents corresponds to having transparent (explicit) narration and opaque content (or semantics). Conversely, formality corresponds to having explicit semantics and opaque narration. In practice, of course, the situation is less definitive. Rigorous mathematical documents (the standard in day-to-day mathematics), make a significant part of their semantics explicit by marking up statements such as theorems or definitions and by using standard notations for representing formulae. Similarly, in the formal world, it is common to give defined symbols human-readable notations. Moreover, narrative-inspired constructs such as sections, lemmas, or proofs are used partly to increase intelligibility for human readers. In both cases various structuring mechanisms are used to organize knowledge.

Therefore, a flexiformal representation language should allow co-representing both the narration and content aspects of mathematical knowledge in a structure preserving way while supporting

partial representations for either aspect.

Then, we use the term *flexiformalization* for converting mathematical knowledge (which we see as inherently flexiformal) into such a flexiformal language. It is important to note that, although fully explicit representations for both aspects are ideal, this rarely happens in practice. However, many applications can still makes use of partial knowledge while also benefiting from more explicit representations.

The Figure on the right shows this for some common applications from the formal world. Search and change management can already be useful with very partial semantics while proof checking and proof search require almost completely formal libraries to be practical. Note that the graph only takes into account the level of formality and not the implementation challenges inherent to each service. For instance, proof checking requires a fully formal theorem and proof while proof search, although conceptually more complex, only requires a fully formal theorem so is less demanding in terms of formality. This scaling of the functionality of applications has two crucial consequences for management of flexiformal knowledge. First, it lowers the cost of converting knowledge (flexiformalization) before a practical gain for the user (due to applications) is achieved. Second, it incentivizes further flexiformalization as it improves the functionality of existing applications and, possibly, makes new ones practical.



Within this context, in this thesis, we conduct an initial, comprehensive case study of flexiformality. The main objectives we pursue are:

- *representing flexiformal knowledge:* survey existing formal and informal mathematical knowledge to observe how flexiformality manifests itself in practice then design and implement a formal language for representing flexiformal knowledge.
- *flexiformalization:* import and integrate existing mathematical archives at different levels of formality into this language to build a large library of flexiformal mathematics and, in the process, evaluate the expressivity of the language and its implementation.
- *flexiformal knowledge management:* design and implement a generic system that incorporates concrete practical applications for using and managing this flexiformal library. Therefore, we evaluate the generality of the language and the scalability of the implementation with respect to building generic, human-facing applications.

This thesis is organized as follows. In Chapter 2 we examine how flexiformality manifests itself in mathematical practice within both formal and informal mathematics and develop a set of phenomena and requirements for a flexiformal representation language. Then, in Chapter 3, we analyze and discuss related languages and systems with respect to coverage of the phenomena and meeting the requirements. Based on that, and building on MMT [RK13a] and OMDOC [Koh06]

which we discuss in Chapter 4, we design and implement a flexiformal representation language for mathematics in Chapter 5. The result, the iMMT language and system, extends MMT with the goal of becoming a join generalization of MMT and OMDoc. We evaluate the resulting language in Chapter 6 by examining how each flexiformal phenomenon can be represented in iMMT. Finally, we develop a practical, flexiformal ecosystem by building a large, heterogeneous library of flexiformal mathematics and a system integrating services and applications for it. Specifically, in Chapter 7 we import and combine knowledge from three existing mathematical libraries, namely the Mizar Mathematical Library [TB85, MML], the SMGloM library [GIJ+15], and the OEIS library [OEI15]. Then, in Chapter 8 we develop several services and applications for flexiformal knowledge that we aggregate into a concrete system. We discuss future work and conclude in Chapter 9.

Part I

Flexiformal Phenomena and State of the Art

Chapter 2

Flexiformal Phenomena in Mathematics

2.1 Phenomena of Mathematical Vernacular

Acknowledgement. *The results in this Chapter are based on collaborative work with Michael Kohlhase that was published in [KI14]. However, the work is significantly revised and updated.*

Common Mathematical Language (CML) is the specialized sublanguage of natural language used for expressing mathematical knowledge. To achieve this goal, CML has developed particular linguistic devices that make it more suitable for expressing mathematical thoughts and knowledge. These include grammatical constructs, idioms and linguistic conventions but also structural adaptations with significant effects on the fundamental properties of the language. The integration of formulae as well as the use of symbolism and complex notations make mathematical discourse more concise and readable for experts while at the same time making it seemingly impenetrable for neophytes. Note that CML includes the ability to extend its own vocabulary, so once readers know enough CML, they can bootstrap and understand any new piece of CML on the basis of prior knowledge.

From the perspective of natural language understanding, this means that CML raises very particular challenges and as a consequence, standard processing tools designed for general natural language often do badly when applied to mathematics [Wol12]. To understand why that is, one must first analyze the linguistic phenomena that are specific to mathematical discourse.

In this thesis, we do not want to give a complete account of the particular syntactic and semantic phenomena of CML – we refer the reader to the work reported in [Bau99, Zin04, Wol13, Gan13] – but we want to highlight some issues that pose challenges to the target formats of semantics construction. This perspective has not been in the focus of the literature so far as most of the work concentrated on semantics construction or extraction algorithms and the particular logical systems tailored to them. So let us start with a high-level assessment of CML before we go into particular challenges in the subsections.

From a high level view, mathematical texts discuss mathematical objects and their relations us-

ing textual representations[1]. Critical parts of assertions or proofs are often reduced to writing down or manipulating syntactic representations of mathematical concepts. At the same time, mathematical documents are written with humans in mind as the target audience so the harsh formalism of mathematical notions is often softened by a carefully constructed narrative structure, examples, intuitions and high-level overviews.

This means that CML must satisfy two conflicting requirements. Firstly, the intrinsic properties of mathematics require mathematical language to be rigorous and precise. Secondly, human authors and readers aim for concise and intuitive results that are easy to write and understand. In fact in many cases the value of proofs, for instance, is measured by their subjectively perceived beauty rather than their precision or even correctness.

Therefore, in practice, the form of each narrative realization of mathematical results depends on notational, didactic and linguistic considerations. It is an important observation that mathematical texts can only be understood as a projection of abstract mathematical results. This is of course true for any language as it is just a code for encoding information. However, in mathematics, the background knowledge is crucial because mathematical texts usually require a more active reader than other text forms.

In the interest of conciseness, mathematical discourse may carry that to an extreme. In practice, ambiguities are accepted if they can be resolved from the context. The correct meaning is left to be inferred by the reader. The understanding process often draws on background knowledge of concepts and notations. It can also involve non-trivial inferences for filling in missing proof steps. Moreover, CML uses symbols, concepts and notations introduced somewhere else (sometimes later) in the text. Even for expert human readers, this is a very involved task and, therefore, it raises significant challenges for computers.

In the rest of this Chapter, we will look at some key language phenomena that are specific to mathematical discourse and differentiate it from generic natural language. We will distinguish phenomena at three levels of granularity: representations of mathematical objects at the phrase level, mathematical statements at the discourse level, and context phenomena at the document or collection levels.

We systematically use examples from a variety of sources to demonstrate and substantiate the phenomena. We use established books and manuals as representatives for the central branches of mathematics as well as a random selection of (around 30) mathematical papers from the ARXIV. For each example we cite the source from where it is taken, except when the example is trivial or it is taken from another compilation of examples and we do not know the original source – i.e. from [KI14]. Note that we select each example to be idiomatic, and use a wide variety of sources simply for coverage. Otherwise, each of the documents investigated alone is sufficient to find examples for most phenomena discussed below.

[1]For the purposes of this thesis we neglect the fact that CML also uses tables, plots, charts, and diagrams besides the textual modality – see, for instance, [Nel97] – and leave their study to future work.

2.1.1 Phrase Structure

P1: Formulae as Primitive Objects Formulae are an essential part of mathematics and, consequently, are also an important part of mathematical discourse. We identify three distinct subphenomena *symbol applications*, *naming objects locally*, and *referential meaning* which we analyze individually. Note that we treat the integration of formulae into narrative text as a separate phenomenon and discuss it in **P2: Mixing Text and Formulae** below.

P1a: Symbol Application One of the most common and basic usages of mathematical symbols and concepts in mathematical formulae is applying them to construct new expressions. Applying symbols such as operators or functions to other symbols such as constants, variables or values occur either in formulae or as part of narrative text. For instance the formula in Example 2.1 can also be expressed narratively as "the sum of a and b".

(2.1) $a + b$

(2.2) $|x - y| \leq \delta \Rightarrow |f(x) - f(y)| \leq \epsilon$ *[Rud76]*

For more complex examples such as (2.2) the narrative representation becomes awkward. Generally, formula notation improves conciseness and readability [Wol13] which makes it ubiquitous in mathematical discourse.

P1b: Naming Objects Locally Symbolic names provide a concise textual representation for mathematical objects in CML. Being, in essence, atomic formulae, they can be used both in formulae and in text. Since mathematical practice relies heavily on syntactic manipulations of formulae, naming objects is essential and, in practice, common.

There are various forms of declarations in CML: (2.3) shows occurrences of binders (here the quantifiers $\forall$, $\exists$) in a (first-order logic) formula. Here the binders merely declare the identifiers ϵ, δ, x, and y, whose scope ranges over the body of the binders (the part after the "."). (2.4) shows a CML version of (2.3), where the binders are verbalized and the declarations contain *domain restrictions* of various forms, e.g. $\delta \in \mathbb{R}^+$ and $\epsilon > 0$. We think of the first as a *type declaration* (aka. sortal restriction in linguistics) and the second as a *propositional restriction*; we distinguish them as they are usually handled differently in reasoning: types are built into the substitution mechanism (type checking, usually kept implicit), whereas propositional restrictions are subject to the general reasoning process. The declaration "$x, y \in \mathbb{R}$ *with* $|x - y| \leq \delta$" has a compound restriction and is placed after x and y have been used; a rather common practice used to make the syntactic structure of the assertion easier to comprehend. For verbalized binders, the scoping of introduced identifiers is often ambiguous. For instance, in (2.5) and (2.6) the scope of x and, respectively, n extends to the second sentence. In the latter example, the fact that the construction is that of a "donkey sentence" [KR93] suggests a dynamic treatment.

(2.3) $\forall\epsilon.\exists\delta.\forall x, y.|x - y| \leq \delta \Rightarrow |f(x) - f(y)| \leq \epsilon$ *[Rud76]*

(2.4) *For all* $\epsilon > 0$*, there is a* $\delta \in \mathbb{R}^+$*, such that* $|f(x) - f(y)| \leq \epsilon$ *for all* $x, y \in \mathbb{R}$ *with* $|x - y| \leq \delta$*.*

(2.5) *Let* $x \in G$. *The element* y *such that* $yx = xy = e$ *is uniquely determined. [Lan10]*

(2.6) *If* $n > 10$ *is prime then it is odd. Moreover* n *ends in one of the digits* $1, 3, 7$ *or* 9.

(2.7) *Define the distance* $d(x, y)$ *between two numbers* x, y *to be* $x - y$. *Show that [. . .] [Lan68]*

(2.7) contains an example of a nested declaration commonly used for relational nouns (which occur often in CML to verbalize functional objects) and their notations. (2.7) introduces the verbalization of the "*distance*" between two numbers together with the notation $d(\cdot, \cdot)$. x and y (the numbers) are the parameters to the definition and are named in a similar way: "*two numbers* x, y" gives the numbers names which can be referenced in the notation (and definition) for the distance.

Note that in all cases the declarations have local scope, i.e. the names are only accessible within the sentence or (more commonly) the containing statement (see Section 2.1.2). Scoping of global names (introduced by definitions) is subject to the document context level (see Section 2.1.3).

P1c: Referential Meaning In natural language, the meaning of lexical entries is grounded in the experience of the interlocutors as embodied agents in the world. Even though it is very difficult to define a concept, e.g. of a chair, in all its aspects, we all share a broad consensus on this matter. In mathematics this is different – possibly because it is difficult to directly experience mathematical objects and concepts. Mathematical concepts and objects are defined in explicit definitional statements (see Section 2.1.2) from previously introduced concepts or objects or declared by selection from non-empty sets with their properties further refined by assumptions or axioms. The only possible exceptions are simple mathematical concepts like $\mathbb{N}$, or $\mathbb{R}^n$, which can be directly experienced by counting and as space. Forgoing a rehash of philosophy of mathematics[2] we subscribe to a referential theory of meaning, where the meaning of lexical items (mathematical symbolism and technical terms) is given by referring to the statements or phrases that introduce them (we call these declarations; see below); see [WG10, WGK11] for linguistic studies that justify this view.

P2: Mixing Text and Formulae What is unique about formulae in CML is their deep integration into narrative text to the point where they became integral parts of the language. Formulae often appear interspersed within the text, e.g. in (2.8). More rarely, text can occur inside formulae, such as in (2.9), and as we see in the slightly more complex (2.10), text and (non-trivial) formulae can recurse freely.

(2.8) *If* A *is atomic, then* $A \notin S$ *or* $\neg A \in S$. *[And02]*

(2.9) $pvar(\alpha) := \{p \mid p$ *occurs in* $\alpha\}$ *[EFT94]*

(2.10) $A \times B = \{Z \mid Z = (X, Y)$ *where* $X \in A$ *and* $Y \in B\}$ *[Coh81]*

Formulae can usually be replaced by their verbalization, i.e. a mathematically equivalent textual representation, typically achieved by reading out the formula as a phrase (see **P3** below). Therefore, linguistically and gramatically, formulae can take on a variety of roles. Here, we abstract

[2] . . . where the jury is still out even on matters like the existence of the natural numbers; see e.g. [BP64]

away from linguistic analysis and processing and refer to [KI14] for a more in depth discussion from that perspective.

P3: Notations and Verbalizations Mathematical objects usually have two concrete textual realizations, *notations* and *verbalizations*. While functionally equivalent, the two fundamentally differ in form and usage. Verbalizations, such as "*plus*" or "*the set of natural numbers*" are natural-language based representations of mathematical notions. They are commonly found in mathematical texts but only rarely inside formulae, usually when there is no standardized, nice-looking notation, e.g. in (2.10). Notations, such as infix $+$ for addition or $\mathbb{N}$ for the set of natural numbers, are associated with the formal representation in mathematical language. They are usually used within formulae although they are also often interspersed throughout the narrative text. For instance in (2.11) "*Euler's number*" is the verbalization and "e" is the notation. A more complex example is shown in (2.12) where both the verbalization "*simple linear group of degree* n *over a field* F" and the notation "$SL(n, F)$" have parameters – in this case the degree n and the field F.

(2.11) *Euler's number,* e *[. . .]*

(2.12) *The simple linear group of degree* n *over a field* F*, denoted* $SL(n, F)$*, is the set of* $n \times n$ *matrices with determinant* 1*, with the group operations of ordinary matrix multiplication and matrix inversion. [Wik]*

Note that from a parsing and linguistic analysis perspective notations introduce an additional challenge due to *ambiguity*. In practice, mathematical notations have implicit and loosely defined scope and precedence which help readers disambiguate formulae. For instance "$\sin x/y - z$" is ambiguous between "$\sin(\frac{x}{y}) - z$", "$\sin(\frac{x}{y-z})$", and "$\frac{\sin(x)}{y-z}$" but, for a reader with the required mathematical background knowledge its meaning is clear (usually because the other readings are nonsensical mathematically). In other cases, ambiguities can be caused by different operators having similar notations, for instance function application and multiplication. However, $f(a(b - c)) = 3$ would be considered clear by most mathematicians due to well established naming conventions, by which f is preferentially used for functions and a and b are used for individuals. For a more in-depth study of ambiguities we refer the readers to [Wol13, Zin04, Gin11, Gan13] and the literature referenced in these.

A special case of notational ambiguity is *elision*. Some operators such as Σ, lim, $\int$ have complex notations where arguments can be omitted and have implicit default values. For instance, $\log n$ is usually taken to be $\log_{10} n$ in most contexts and $\log_2 n$ in others; given a sequence $a_0, a_1, \ldots, a_k$ in the context, Σa_i is automatically understood by an experienced reader as $\Sigma_{i=0}^{k} a_i$. Additionally, the arity of mathematical notations can be flexible as there are symbols that take sequences as arguments. This includes sets, lists, vectors, matrices but can also be extended to most infix binary operators in the presence of associativity.

Since we are focused on the target format rather than the parsing and processing aspect we do not consider ambiguity and elision central concerns. However, being able to represent symbols with multiple, possibly ambiguous notations, as well as those with flexible arity is essential. Moreover, in practice, ambiguity in the source can lead to consequences in the processing result (e.g

multiple parses or incomplete semantics). We cover this with the requirement **R3: Underspecification** below.

2.1.2 Discourse Structure

P4: Mathematical Statements At the discourse level, CML is more explicit in delineating the role of important paragraphs than common natural language. Definitions, theorems, lemmas, corollaries, examples, exercises, or proofs are often marked up – even in informal text – using specific fonts, emphasis or keywords and numbers for referencing. We distinguish the most important cases below as subphenomena and discuss them individually.

P4a: Assertions Under the name of *assertions* we group a variety of mathematical statements that assert something about the truth value of a proposition. This includes *axioms* (e.g 2.13), *theorems* (e.g 2.14), *lemmas* (e.g 2.15) and *corollaries* (e.g 2.16). Assertions are often associated with proofs which typically have more complex internal structure. We discuss proofs as a separate sub-phenomenon below.

(2.13) ***Axiom of extensionality.*** *Two sets are equal (are the same set) if they have the same elements:* $\forall x, y.(\forall z.z \in x \Leftrightarrow z \in y) \Rightarrow x = y$. *[Wik]*

(2.14) ***Femat's Little Theorem.*** *If* p *is a prime number and* a *is a natural number, then* $a^p \equiv a$ *(mod* p*).* *[MWd]*

(2.15) ***Lemma 2.*** *If* $a, b \in \mathbb{Z}$ *and* $b > 0$*, there exist* $q, r \in \mathbb{Z}$ *such that* $a = qb + r$ *with* $0 \leq r < b$. *[IR98]*

(2.16) ***Corollary*** *If* f *is continuous on* $[a, b]$ *and* $f = g'$ *for some function* g*, then* $\int_b^a f = g(b) - g(a)$. *[Spi94]*

Assertions often have specific relations to others statements. For instance, a proof proves an theorem, a lemma contributes to a proof, and a corollary follows a theorem. Furthermore, a corollary implies the proof (from the theorem) is trivial, while a classification as a theorem implies that a proof exists, whether or not it is given in the text.

P4b: Definitions An important particularity of the mathematical vernacular is that it is not fixed. Instead, it is dynamic and can extended both locally within formulae (see **P1b** Naming Objects Locally) as well as globally using *definitions*. They allow mathematicians to declare and name new concepts that can be used later. For instance definition (2.17) declares the set intersection operator $\cap$ that is used later to construct new sets. Similarly, definition (2.18) declares the complex-conjugate function that is used to produce and relate complex numbers.

(2.17) *If* S_1, S_2 *are sets, then the* ***intersection*** *of* S_1 *and* S_2 *denoted by* $S_1 \cap S_2$*, is the set of elements which lie in both* S_1 *and* S_2. *[Lan10]*

(2.18) *The* ***complex conjugate*** *of a complex number* $z = a + ib$ *is defined to be* $\bar{z} = a - ib$. *[MWd]*

(2.19) ***5.1 Definition.*** *Let $\mathfrak{A}$ and $\mathfrak{B}$ be S-structures. [...] $\mathfrak{A}$ and $\mathfrak{B}$ are said to be* ***isomorphic*** *(written $\mathfrak{A} \cong \mathfrak{B}$) if there is an isomorphism $\pi : \mathfrak{A} \cong \mathfrak{B}$.* *[EFT94]*

Definitions lead to a layered construction of mathematical knowledge where ever-more complex concepts are built (defined) based on simpler ones. For instance, definition (2.19) of the predicate *isomorphic* uses the concept of isomorphism which was defined previously. Additionally, is uses the type S-structure which was also defined previously. Defining new *types* is a particularly interesting kind of definition which we treat as a separate sub-phenomenon and discuss it below.

P4c: Type Declarations The notion of objects having *types*, (e.g. "*Complex Number*", "*Polygon*", "*Prime Number*"), is ubiquitous in mathematical practice. Note that we use the term *types* in an unconstrained sense here to simply mean specific, named sets of individuals. But not every type is a set, for instance "***set***" itself. Such types are used to construct and classify mathematical objects. For instance, Example 2.20 introduce the new concept of sets, Example 2.21 defines prime numbers as a subset of naturals, and Example 2.22 constructs a new type (complex numbers) as, effectively, a pair of real numbers.

(2.20) *A collection of objects is called a* ***set*** *[Lan10]*

(2.21) *A* ***prime number*** *(or a* ***prime****) is a natural number greater than 1 that has no positive divisors other than 1 and itself.* *[Wik]*

(2.22) *The* ***complex numbers*** *are the field $\mathbb{C}$ of numbers of the form $x + iy$, where x and y are real numbers and i is the imaginary unit equal to the square root of -1, $\sqrt{-1}$.* *[MWd]*

Also, like other definitions, types are used for clarity and conciseness when introducing new objects. Concretely, such types function as predicates describing the properties of the newly introduced objects which can later be used in proofs involving them. For instance, in Fermat's Little Theorem from Example 2.14 p is introduced as a prime and a as a natural number.

P5: Structured Declarations Structured declarations are special statements that have internal structure and context. We distinguish between several kinds of structured declarations below.

P5a: Inline Statements Even though statements are usually paragraph-level structures, abbreviated forms also appear at the phrase level. Consider the situation in (2.23) where we use the existence and uniqueness of a solution to a (differential) equation to define a new concept (the exponential function) in a parenthetical phrase. This construction is usually considered to be mathematically equivalent to the combination of the explicit statements (2.24) and (2.25).

(2.23) ***Theorem 7.1****: There is exactly one solution (called the* ***exponential function*** *$f(x) = e^x$) to the equation $f' = f$ with $f(0) = 1$.*

(2.24) ***Theorem 7.1****: The equation $f' = f$ has exactly one solution with $f(0) = 1$.*

(2.25) ***Definition 7.2****: We call the function f with $f' = f$ and $f(0) = 1$ the* ***exponential function*** *and write $f(x) = e^x$.*

(2.26) *Let E be a vector space. A* ***norm*** *on E is a function $v \mapsto |v|$ from E into $\mathbb{R}$ satisfying the following axioms:*
N1. *We have $|v| \geq 0$ and $|v| = 0$ if and only if $v = 0$.*
N2. *If $a \in \mathbb{R}$ and $v \in E$, then $|av| = |a| \cdot |v|$.*
N3. *For all $v, w \in E$ we have $|v + w| \leq |v| + |w|$.*
[Lan68]

(2.27) ***Definition 3.17*** *The number e is an important mathematical constant, approximately equal to 2.71828, that is the base of the natural logarithm. It is the limit of $(1 + \frac{1}{n})^n$ as n approaches infinity, an expression that arises in the study of compound interest, and can also be calculated as the sum of the infinite series $e = 1 + \frac{1}{1} + \frac{1}{1\cdot 2} + \frac{1}{1\cdot 2\cdot 3} + \frac{1}{1\cdot 2\cdot 3\cdot 4} + \cdots$* *[Wik]*

Example 2.26 shows a different case where a definition requires three named inline declarations. The three identities are arguably local to the definition unlike the exponential function from Example 2.23 which is global. The last example (2.27) shows that inline statements can be intricately nested to form complex, but concise statement-scapes, featuring a (primary) definiens, alternative definitions, properties, and even hints for calculations.

P5b: Mathematical Structures Mathematical structures (e.g. "*equivalence relation*", "*commutative group*" or "*vector space*") are constructs formed from one or more sets together with elements, operations, and relations on them which must satisfy various properties (sometimes called requirements or axioms).

In CML mathematical structures typically function as types (see **P4c**) but often behave differently due to their internal structure. CML definitions, theorems and proofs involving mathematical structures often work within the structure as if it were a standalone theory and treat its sets, operations and properties as standalone declarations. We call this the *pragmatic* perspective as opposed to the *strict* perspective where mathematical structures are treated as types without focusing on their internals. For instance, the problem in (2.30) asks about the equivalence of two statements within a finite field. Although the problem statement uses the pragmatic abstraction "$K\,[X]$", the natural proof that the two are derivable from each other works within the context of the operations, relations and axioms that make up the field declaration (using an implicit forall introduction to prove case *(b)*). On the other hand, the problem in (2.31) involves abstraction over the set (type) of finite groups as well as the set of morphisms between them. Therefore, mathematical practice makes use of both perspectives and while some problems (e.g. (2.30)) can be expressed and solved using the strict perspective others require the pragmatic one (e.g. (2.31)).

(2.28) *A* ***group*** *is a monoid (§**2**, no. 1, Definition 1) in which every element is invertible. [Bou74]*

(2.29) ***Definition 4.*** *Let $\langle R, +, 0, - \rangle$ be a commutative group, and $\langle R, *, 1 \rangle$ a monoid, such that $\langle R, +, * \rangle$ is a ringoid. Then we call $\langle R, +, 0, -, *, 1 \rangle$ a* **ring***.*

(2.30) ***Problem 1.*** *Let $\mathcal{K}$ be a finite field. Prove that the following statements are equivalent:*

(a) $1 + 1 = 0$*;*
(b) for all $f \in \mathcal{K}[X]$ *with* $deg\ f \geq 1$, $f\left(X^2\right)$ *is reducible.*

[APS]

(2.31) ***Problem 2.*** *Let* $\mathcal{G}$ *be the set of all finite groups with at least two elements.*
(a) Prove that if $G \in \mathcal{G}$*, then the number of morphisms* $f : G \to G$ *is at most* $\sqrt[p]{n^n}$*, where* p *is the largest prime divisor of* n*, and* n *is the number of elements in* G*.*
(b) Find all the groups in $\mathcal{G}$ *for which the inequality at point a) is an equality.*

[APS]

P5c: Proofs Reasoning about the truth of propositions is, arguably, the primary concern of mathematics. Therefore, proofs are one of the central and distinguishing features of mathematical documents. They consist of text fragments of various sizes – ranging from single words as in (2.32) over meta-instructions as in (2.33) to complex argument structures that can span a whole chapter or even an entire book[3]. But, even though proofs sometimes straddle the three levels we used to structure our phenomena, they are most commonly structured discourse elements as in (2.36).

(2.32) ***Proof****: Trivial* □

(2.33) ***Proof****: The proof is left to the reader as an exercise.* □

(2.34) *. . .* $f(n)$*, which must obviously be positive.*

(2.35) *. . . thus we have . . . , hence we have proven the assertion.*

(2.36) *Assume there are only finitely many primes. Let* p *be the largest prime number and* P *be the product of all primes less than or equal to* p*; then consider the number* $P + 1$*:*

$$P + 1 = (\Pi_{q \leq p} q) + 1.$$

Two cases are possible: either (a) $P + 1$ *is prime, or (b)* $P + 1$ *is not prime. But if (a) is true,* $P + 1$ *would be a prime number larger than* p*. And if (b) holds, none of the primes* $q \leq p$ *is a prime factor of* $P + 1$*, so the prime factors of* $P + 1$ *are all larger than* p*. In both cases, the assumption that there is a largest prime* p *leads to a contradiction. This shows that there must be infinitely many primes. [Rib96]*

Usually proofs are structured into proof steps, which consist of an (intermediary) assertion with a justification. In larger proofs, proof steps can be interspersed by proper statements like definitions, assertions, statement of subgoals, subproofs, etc. The relation between all of these are given by the justifications, which can be explicit, lexicalized by special termini like "*thus*" and "*hence*" in (2.35), or (often) elided or implicit as in (2.32) and (2.33). The internal structure

[3]The proof of the classification of finite simple groups, which consists of tens of thousands of pages in several hundred journal articles written by about 100 authors, published mostly between 1955 and 2004 is a somewhat extreme example.

mediates complex scoping structures that determine the visibility of identifiers/names and the accessibility of results.

Proofs are one of the more interesting and well-studied linguistic phenomena, and we refer to the discussions in [Wol13, Ker10, Avi08, Mel93] for more in-depth analyses.

2.1.3 Document Context

P6: Modularity Mathematical knowledge is written down and disseminated in the form of individual documents such as books, articles or lecture notes. At the same time there is an orthogonal, pragmatic organization of mathematical knowledge branches, fields or theories of mathematics (e.g. "algebra", "number theory", "mathematical logic", etc.). We therefore distinguish two forms of modularity corresponding to documents and, respectively, mathematical theories and discuss them below. Note that we explore how such *modules* are related and reused separately in **P7: Framing** and, respectively, **P8: Common Ground**.

P6a: Grouping Statements Mathematical documents encapsulate knowledge in the form of statements (see **P4** and **P5**) such as definitions, theorems, notations, proofs, etc. Within documents, there can be paragraphs, sections, or chapters which provide additional encapsulation and influence scoping of declarations and variables. These containers of knowledge can be and often are reused in various forms in other documents.

P6b: Dynamic Theories In mathematical practice, *theories* tend to be very large and fluid rather than fixed. First, instead of documents, they corresponding to entire mathematical branches or fields such group theory, number theory, etc. Second, they are implicitly enriched with new results when they appear or are used, rather than being explicitly declared in an immutable document. Review and survey articles are sometimes produced to provide an explicit snapshot of some mathematical theory at a point in time. We call such theories as used in the context of mathematical practice *dynamic theories* as opposed to the static groups of statements typically used in representation languages (and mentioned above in **P6a**).

Mathematical theories exist in this form because mathematics is focused on usability in practice. Mathematicians want an extensive tool-chest of concepts and theorems to develop new proofs and results. However, from a knowledge management perspective, theories that are small and fixed are easier to declare, organize and reason about. See, for instance, the theory-graph-based little-theories [FGT92] approach. This makes dynamic theories as a phenomenon particularly interesting to represent adequately.

P7: Framing An important property of mathematical thought is *framing* certain notions in terms of others. In mathematical discourse objects of interest are often viewed in terms of already understood structures and creative use is made of this perspective. This allows relating seemingly distinct mathematical fields as well as establishing new results; see [KK09, Koh14b] for a discussion.

For example, the set of integers can be viewed as a group under addition, subsets of a fixed set

can be seen as (characteristic) functions and functions can be seen as sets (of input/output pairs). More advanced results include interpreting a Boolean algebra as a field of sets via Stone's representation theorem or embedding an abstract manifold into some Euclidean space using Whitney's embedding theorems.

Framing is deeply ingrained in mathematical thought and naturally transpires into the presentation of mathematical results. Consider for instance (2.37) and in particular the adjective "*discrete*", which applies to metric or topological spaces[4]. But it is well-known that a normed vector space can be viewed as a metric space: $d(x, y) := \|x - y\|$ is a metric. So we can felicitously utter (2.37) to any interlocutor who can frame a normed vector space as a metric space. (2.38) is similar, only that the concept of a "*ball*" is a concept in metric spaces and the concept of "*open*" sets in topological spaces – here we have a case of "nested framing" and need the additional framing that metrics induce topologies.

(2.37) *A normed vector space* $(V, \|\cdot\|)$ *cannot be discrete.*

(2.38) $|\cdot|$ *is a norm on* $\mathbb{R}$ *. . . Let* $B \subseteq \mathbb{R}$ *be an open ball, . . .*

Mathematical adjectives (and nouns in declarations) are notoriously prone to be applied via frames – see also [Koh14a], so framing appears as a important challenge in the understanding and semantic representation of mathematical texts.

P8: Common Ground Mathematics is a heterogeneous field and mathematical texts in different areas routinely use distinctive notations and vocabularies. This is a natural consequence of the modular nature of mathematical practice and of mathematics' dynamic vocabulary. In practice, mathematical texts remain understandable due to a complex notion of document context that *resembles an object-oriented model*. As mathematical documents build on existing knowledge, they implicitly *inherit* concepts, notations and verbalizations from such sources to create a specific mathematical vocabulary. We call this process "establishing *common ground*" and, in practice, it is partly explicit via recaps of inherited notions and partly implicit leading to ambiguity. The former typically appears in a preliminaries section that briefly restates relevant developments while possibly overriding some definitions and notations. The latter is typically used for foundational and logical assumptions which are rarely made explicit in mathematical papers and documents and instead are left for the reader to determine. We discuss each as individual sub-phenomena below.

P8a: Recaps Standalone mathematical documents such as articles or books have a standard narrative structure: an introduction, followed by the main part that presents the body of knowledge the document is written to convey (the payload of the document), which is followed by conclusions. Often, such documents also contain a presentation of related work and a recapitulation of the conceptual foundations to make the document self-contained and/or introduce the concepts and notations needed to understand the payload – for lack of a better word we call such document fragments *recaps*.

[4]stating that the metric or topology is discrete (i.e. separates points), which a norm cannot do, since it has to scale with scalar multiplication (this is the core of the assertion in (2.37)).

The snippet from Example 2.39 is taken from [HK15] and defines the cover of the multiplicative group of an algebraically closed field. However, later, the authors source the concept origin to an earlier paper ([Zil03] – "*[13]*" in the text) and effectively import the terminology, definitions and theorems. For instance, when establishing results, [HK15] mentions "*Moreover, with an additional axiom (in $L_{\omega_1\omega}$) stating $K \cong Z$, the class is categorical in uncountable cardinalities. This was originally proved in [13] [. . .]. Throughout this article, we will make the assumption $K \cong Z$.*". Therefore it directly builds on results from another paper.

The situation from Example 2.40 is more complicated as it builds on developments from different papers that use equivalent definitions. The snippet shows a definition introducing multinets from a paper ([Bar15]) studying their properties. Later in that section some concrete properties of multinets are introduced with the phrase "*Several important properties of multinets are listed below which have been collected from [4,10,12].*". The referenced papers all use slightly different definitions of multinets but they are assumed to be equivalent so that the properties hold. In fact, in this paper the assumption is made explicit – although not proved – from the start: "*There are several equivalent ways to define multinets. Here we present them using pencils of plane curves.*"

Example 2.41 is from a paper ([CS09]) that studies the halting problem for accelerated Turing machines. The paper gives a telegraphic version of the full definition, which is given in the literature. Actually [CS09] continues with an overview of the literature, citing no less than 12 papers, which address the topic of accelerated Turing machines. One of these supposedly contains the formal definition, which involves generalizing Turing machines to timed ones, introducing computational time structures, and singling out accelerating ones.

(2.39) ***Definition 1.1*** *Let V be a vector space over Q and let F be an algebraically closed field of characteristic 0. A* cover of the multiplicative group *of F is a structure represented by an exact sequence $0 \to K \to V \to F \to 1$, where the map $V \to F^*$ is a surjective group homomorphism from $(V, +)$ onto $(F, \cdot)$ with kernel K. We will call this map* exp*. [HK15]*

(2.40) ***Definition 2.1*** *The union of all completely reducible fibers (with a fixed partition into fibers, also called blocks) of a Ceva pencil of degree d is called a $(k, d) - multinet$ where k is the number of the blocks. The base X of the pencil is determined by the multinet structure and called the base of the multinet. [Bar15]*

(2.41) *An **accelerated Turing machine** (sometimes called **Zeno machine**) is a Turing machine that takes 2^{-n} units of time (say seconds) to perform its n^{th} step; we assume that steps are in some sense identical except for the time taken for their execution. [CS09]*

(2.42) ***1.5 Topological spaces*** *A* topological space *is a set S in which a collection τ of subsets (called* open sets*) has been specified, with the following properties: S is open, $\emptyset$ is open, [. . .] Such a collection is called a* topology *on S. [Rud73]*

(2.43) ***1.6 Topological vector spaces*** *Suppose τ is a topology on a vector space X such that*

> *(a)*every point of X is a closed set, and
> *(b)*the vector space operations are continuous with respect to τ
> *Under these conditions,* τ *is said to be a* vector topology *on* X*, and* X *is a* topological vector space*.* *[Rud73]*

The situation in mathematical textbooks is similar in structure to that in research papers – perhaps more pronounced. Consider the passages in (2.42) and (2.43) taken from Rudin's classical introductory textbook to Functional Analysis [Rud73]. Rudin does not directly cite the literature in either of these definitions. But in the preface he mentions the vast literature on function analysis and in Appendix B he cites the original literature for each chapter. The situation in textbooks is also different from research articles in that textbooks – like survey articles, and by their very nature – do not add (much) new knowledge or results, but aggregate and organize the already published ones, possibly reformulating them for a more uniform exposition. But still, one can distinguish recap parts – as the ones above – which are much more telegraphic in nature from the primary material presented in the textbook.

The common part in all examples above is that the document establishes the *realm* of mathematical knowledge to which the current development belongs (e.g. multinets, accelerated Turing machines or functional analysis). Therefore, it situates the content of the document within the mathematical knowledge space. This is not only relevant for understanding the document by establishing a common ground. It also outlines the context in which the document and its payload contributes back to the existing mathematical knowledge.

Additionally, as shown in the examples above, mathematical documents typically rephrase or reframe concepts and notations when establishing common ground. Therefore, concepts are not always directly imported but often are modified via renamings and redefinitions or by giving local notations and verbalizations.

P8b: Foundational Ambiguity Mathematical documents can (and typically do) leave foundational and logical assumptions implicit. This can lead to ambiguity when interpreting the contents of the document. For example, mathematical discourse makes use of a large set of concepts whose existence is implicitly justified by them being definable from some foundational primitives. Notable examples are *natural numbers* and *tuples* which, in more formal developments, are usually defined based on sets. Another common example is sequences which also come with their own notational infrastructure, for instance elision (see **P3**).

The reader, if interested in such things at all, is expected to determine what are the compatible foundations in which the document can be interpreted. This is useful because it leaves the document payload open to any foundation in which the needed assumptions and concepts can be derived and, respectively, defined. It is also particularly interesting because the individual foundational options can be incompatible with each other. For instance, the problem of whether a finite game is determined (i.e. there is a winning strategy) can be proved in both a foundation with axiom of choice and in one with the axiom of determinacy. But the two are known to be incompatible [MS62].

Therefore, the challenge with respect to representing mathematical knowledge is not just the

multitude of alternative logics and foundations. It is also the fact that mathematical practice often intentionally avoids foundational commitments.

2.1.4 Level-Independent Phenomena

Below we discuss phenomena that straddle the three levels we distinguished above.

P9: Structural Dimorphism Mathematical knowledge has a dual role: to be precise and rigorous – close to the semantics – on the one hand and concise and intuitive – understandable by humans – on the other. Arguably, it is these irreconcilable requirements that give rise to two kinds of representations usually associated with formal and, respectively, informal mathematics. While these two kinds of representation are meant to refer to the same platonic world (the world of mathematics) it has proven difficult to formally express this connection. One can easily relate a formal document with its informal counterpart or a formal proof with its informal version but the relations are more complex and fine-grained than that. We observe that the narrative and the content representations often have different internal structure.

This practical and structural difference which we call *structural dimorphism*[5] occurs at all levels identified above. At the phrase level this is most transparent with CML itself using and mixing what is arguably two languages for representing formulas. The first is natural language such as in Example 2.44 and the second is mathematical formula language such as in Example 2.45. In fact the two examples convey the same mathematical object, but with different representations. A similar situation occurs above in Examples 2.4 (for natural language) and 2.3 (for formula language) above. Example 2.46 shows how the flexibility of natural language allows a different order or structure from the formula representation. In this case the binding happens at the end (often introduced by the keyword `where`). Similarly, Example 2.5 above shows one formula split over several sentences.

(2.44) *For all* $a, b \in \mathbb{N}$ *there exists a number* m *such that* $a|m$ *and* $b|m$.

(2.45) $\forall a, b \in \mathbb{N}.\ \exists m.\ a|m \wedge b|m$.

(2.46) *There exists a number* m*, such that* $a|m$ *and* $b|m$ *for any* $a, b \in \mathbb{N}$.

(2.47) ***Theorem 7.1****: There is exactly one solution (called the* ***exponential function*** $f(x) = e^x$*) to the equation* $f' = f$ *with* $f(0) = 1$.

(2.48) ***Definition 1.*** *Let* X *be a set, then* $\{\emptyset, X\}$ *is a topology on* X*, it is called the* ***trivial topology*** *and* $\langle X, \{\emptyset, X\}\rangle$ *the* ***indiscrete space*** *over* X.

We observe that discourse-level declarations also commonly have internal structure that differs from the structure of their semantics. Consider, for instance, the hybrid of a definition and a theorem from Example 2.47 (a repeat of Example 2.23). This kind of "definition" is commonly used in informal mathematics and pragmatically combines two actions into one aggregated form:

[5] cf. the usage in crystallography the phenomenon that some substances have two chemically identical but crystallographically distinct forms.

i) the assertion that there is an unique function f such that $f' = f$ and $f(0) = 1$ and *ii*) defining the *exponential function* as the unique function implied by the assertion. Therefore, what is narratively one declaration, amounts to two related declarations from the content (semantic) perspective. A similar situation occurs in example 2.48 where there is again one narrative declaration corresponding to several from a content perspective: *i*) the assertion that $\{\emptyset, X\}$ is a topology on X, *ii*) defining the *trivial topology* as $\{\emptyset, X\}$ and *iii*) defining the *indiscrete space* over X as $\langle X, \{\emptyset, X\}\rangle$.

We observe the same phenomenon at the document context level. Mathematical documents use structuring constructs such as *chapters*, *sections*, *paragraphs* or *sentences*. But such structures are usually mostly narrative and do not directly determine the context and scoping within the document. The proof from Example 2.36 above shows that in the context of a proof where the scope of local declarations is split over several sentences.

We believe that any representation language for mathematics must take seriously the narrative-content duality of mathematical texts. This is practically necessary so that knowledge in that language should be at the same time understandable by humans and processable by machines. Therefore, we emphasize the importance of both content and narrative aspects as well as integrating between them.

2.1.5 Additional Requirements

In this Section, we develop some practical requirements for flexiformal representation languages that come from management, efficiency, and interoperability considerations. They are induced by the observation that the linguistic study of mathematical documents is a niche subject, where researchers have to collaborate to be able to create the necessary linguistic resources like gazetteers, semantic lexica, etc. This leads to the following three (distinct, but interrelated) requirements:

R1: Opaqueness Mathematics is often considered precise and rigorous with the formalization challenges coming from the complexity and ambiguity of CML rather than from mathematics itself. As some of the phenomena above show (e.g **P2** Mixing Text and Formulae, **P8b** Foundational Ambiguity, **P9** Structural Dimorphism), we disagree with that perspective. But even assuming it, a language for representing mathematical knowledge should be applicable to imprecise or incomplete knowledge. This is because (math-specific) natural language processing tools, which are often the sources for such knowledge, produce partially semanticized documents. There, only some parts or aspects of the document semantics are explicit, while others parts remain *semantically opaque* – i.e. the meaning is inaccessible for current computational systems. There is a prevalence, in the real world, of informal documents and of such efforts to semanticize them [SK08]. Even in the formal world, exports for systems can be partial. For instance the MIZAR export (discussed in more detail in Section 7.1 below) skips proof steps that are inferred by the system. Moreover, imports to a different formal language may be unable to fully preserve the original semantics. Therefore, being able to represent such partially semanticized documents is essential.

R2: System Scalability Representation languages must balance expressiveness and minimality. The former increases coverage (e.g. of the phenomena discussed above) but the latter simplifies implementing and reasoning about the language. For a complex language, even if an implementation is realizable, it can be difficult to re-purpose or scale the system to new applications. Common mathematical language itself is an extreme example of a language with extensive coverage but which lacks minimality. As a result of its complexity it is yet to be given a formal semantics let alone an implementation. In this thesis we consider realizing machine-driven practical applications as a central goal, alongside comprehensive mathematical knowledge representation. Therefore we view minimality, with the focus on system scalability, as an important requirement.

R3: Underspecification Under the name "underspecification" the natural language semantics community discusses a set of techniques for dealing with ambiguity by deliberately omitting information from linguistic descriptions to capture several alternative realizations of a linguistic phenomenon in one single representation and how to process such underspecified representations without multiplying them out. Approaches include lexical techniques for polysemous words [Pus98], packed formula representations [DT00] and meta-languages that describe a set of formulae [KTP10]. Even though mathematical language is often regarded as virtually unambiguous, semantic target formats should support underspecified representations. This is particularly important for the lexical level, where polysemy is rampant.

2.2 Narration and Structure in Formal Mathematics

The field of formal mathematics includes a variety of different languages and systems. While the principal goal of such languages is the adequate representation of the semantics of mathematical knowledge, there are also components that are inspired from or reminiscent of informal mathematics. In fact, narrative phenomena abound in formal mathematics also. Much of the information encoded into formal mathematical libraries is not strictly necessary for proof checking. Examples include documentation, notations, statement roles (e.g. lemma vs theorem), structured declarations and proofs, document structure and sectioning information as well as the pragmatic layer designed for human authors and readers. All could be removed or de-constructed to long lists of logical inferences while preserving the provability and formal semantics. But, as emphasized in the discussion on common mathematical language, narration and structure are, alongside semantics, essential components of mathematical knowledge. Therefore, it is unsurprising that they are found in formal systems and libraries also. For the purposes of this thesis we are also interested in a language that is capable of representing the narration and structure as it manifests in libraries of formal systems not just in informal texts.

Therefore, in this Section, we analyze these language features specifically, from the perspective of the phenomena developed in the previous section. There are large number of formal systems which all differ in terms of syntax, choice of foundation or implementation. Notable systems include Mizar [TB85], Isabelle [NPW02], Coq [BC04, Coq96], HOL Light [Har11], Agda [BDN09], Matita [ARCT11], Lean [dMKA$^+$15] and the IMPS system [FGT93]. We will

mostly look at Mizar, Isabelle, Coq, and MMT [RK13a] as idiomatic examples and refer to [RK13a] for a full overview from the purely formal perspective.

P1 Formulae as Primitive Objects At the formula level, different formal systems provide different primitives depending on their base logic and foundation. However, the fundamental constructs used for building formulas using those primitives can be distilled to the same ones as in CML. The basic constructs are symbol application, binding as well as referencing symbols and variables. These are also, effectively, the central primitives used by formula representation languages such as OPENMATH [BCC+04] and Content MATHML [CIM14].

P2 Mixing Text and Formulae Similar to programming languages, *comments* as user-readable annotations of formal declarations and proofs represent a relatively common language construct among formal systems. In the simplest case, comments represent arbitrary text delineated by dedicated syntactic markers that is typically ignored by the system implementing the language. We call such comments *throwaway*. However, in some cases languages provide dedicated syntax for comments to talk about or refer to formal objects. Moreover, like in programming languages, comments are sometimes used by document work-flows other than proof checking, such as generating documentation or presentation. Isabelle, has source comments (which are plain and throwaway) as well as formal comments (which are more structured) that are used for instance, when generating PDF articles as presentation of Isabelle theories. The structuring mechanism in formal comments is "antiquotation", which allows comments to talk about formal objects by recursing back into the object language (with the keyword "@"), effectively permitting a mix of formal and informal content (see Example 1). Moreover Isabelle has both inline comments (separated using "`--`") and declaration-level comments (separated by "`text`").

Example 1. The listing below shows a lemma with inline comments from [AFP][6]. The two comments, while separate, represent one statement that is split because of code-style considerations (line width). The first comment makes use of antiquotation to embed an object into the informal part in order to refer to the "`split_if`" lemma. While this is a fairly trivial example of antiquotation, it can be used to embed arbitrarily complex objects, including ones that recurse back to the comment language.

```
1 lemma cases_simp: "((P --> Q) & (~P --> Q)) = Q"
2 -- {* Avoids duplication of subgoals after @{text split_if}, when the true
     and false *}
3 -- {* cases boil down to the same thing. *}
4 by blast
```

Mizar, has only one type of comments (separated using "`::`") which are plain text and are ignored by the proof checker (as expected) and the PDF presenter but are used by the HTML presenter.

Example 2. The listing below shows a theorem from the Mizar Mathematical Library [MML][7]

[6] at `http://isabelle.in.tum.de/library/HOL/HOL/HOL.html`
[7] at `http://mizar.org/version/current/mml/xboole_1.miz`

```
1 ::$N Modus Barbara
2 theorem
3 X c= Y & Y c= Z implies X c= Z;
```

Similarly, in Coq comments are plain and throwaway, as they are discarded by the parser and treated as blanks.

P3 Notations and Verbalizations Formal systems do not use verbalizations since they are, for the most part, exclusively focused on the formal aspect. However, they do have notation declarations in various forms. For instance, in Mizar notations are usually part of the definition itself as in the listing below.

```
1 func X \/ Y -> set means ...
```

Mizar also has `notation` as a standalone declaration which can be used to declare synonymic and antonymic notations for preexisting symbols. Isabelle, provides primitives in the syntax for the usual fixities including `infix` and `binder`. But, abstracting away the concrete syntax, Coq, Isabelle, Mizar and MMT notations are all mixfix notations with placeholders for arguments (or variables for binders).

P4 Mathematical Statements At the statement level most formal systems provide a relatively rich set of CML-inspired language constructs for representing mathematics.

For instance Mizar has `definition`, `theorem` and `notation` all with the obvious role. In addition Mizar also has `reservation` which reserves variable names for certain types. This is typical although implicit in math, where variable names have often implicit type associations in every field (e.g. m, n are usually used for natural numbers, p for primes, etc.). We discuss the Mizar statement-level constructs in more detail in Section 7.1.

The situation is similar for Isabelle and Coq which have constructs such as `definition`, `lemma`, `notation`, `proof` and `theorem`. LCF-style provers (e.g. the HOL family) use the programming language for statements. MMT on the other hand, provides a single language construct for declaring plain statements: `constant`. Rather than being primitive, the role may be annotated to a constant declaration as metadata. However, in general, formal systems do have a considerable taxonomy of statement level constructs with CML-inspired meaning.

P5 Structured Declarations Even though not always strictly needed by the parser or proof checker, structure in declarations makes them easier to process (understand or create) by humans. Therefore, as in CML, statement-level declarations with internal structures are common in formal systems also. They are most commonly used for mathematical structures (typically called inductive types) and for structured proofs.

For the former, Mizar has a dedicated language construct `structure` which aggregate declarations and provide selectors for the fields. We discuss Mizar structures in more detail in Section

7.1. Isabelle uses *records* for the same purpose, which can extend other records and have select and update functionality [WBB+]. In some cases module-level elements (see modularity and framing below) are used instead for representing such structured declarations. For instance Coq uses *modules* to develop algebraic structures in their Standard Library [Coq96, Chr03] (see also **P6: Modularity** below). Similarly, in the MMT library LATIN [CHK+11], algebraic structures are declared as theories and relating using imports and views.

Proofs are another instance of structured declarations in formal systems. Languages include constructs for *local constants* and *assumptions* that will be used to produce the final proof later. Moreover, there are several *proof methods* that are distinguishable in their importance in mathematical discourse such as induction, proof by cases, proof by negation etc. Logically they can be seen as plain proof rules or tactics but some formal languages distinguish them at least in their syntax. For instance, *proof by cases* is a primitive in Mizar alongside local constants and assumptions (see listing below). Other languages such as Coq use an extensible set of proof *tactics* with a similar effect.

```
1 unionE  : ded A in (X union Y) -> ded A in X or A in Y =
2   [p] existsE (bigunionE p)
3   ([x][q: ded x in uopair X Y and A in x] orE (uopairE (andEl q))
4     ([r: ded x eq X] orIl (congEl r ([u] A in u) (andEr q)))
5     ([r: ded x eq Y] orIr (congEl r ([u] A in u) (andEr q)))
6   ).
```

P6 Modularity Formal systems typically have one or more constructs for grouping statements and managing their scope. At a minimum, they provide an equivalent to mathematical documents to practically organize large libraries into separate files. These include *articles* in Mizar, *theories* in Isabelle, *documents* in MMT and *libraries* in Coq. Many systems, including Mizar, Isabelle and Coq also provide some form of narrative-inspired sectioning mechanism. The *section* construct may have some additional semantics. For instance, in Mizar, *sections*, introduced with the keyword `begin` determine the scope of lemmas. In Coq they manage the scope of local *variable* declarations. Such declarations behave like normal constants from within the section but, from the outside, they are abstracted over using binders where they are used.

Other, more fine grained, examples of constructs for declaring logical modules include *locales* in Isabelle [KW99], *modules* in Coq, and *theories* in MMT. They are named, possibly parametric (e.g. in Coq and Isabelle) lists of declarations that come with an infrastructure for relating and reusing them. We refer to [RK13a, Sect. 2.1&4] for a more detailed overview of modularity in formal systems.

P7 Framing In formal systems some framing-like notion typically exists to relate the logical modules discussed above (e.g. the theories in MMT, locales in Isabelle, modules in Coq, etc.). In MMT, *views* are translations between two theories demonstrating one as being a realization of the other. MMT views map symbols in the source (specification) theory to expressions in the target (realization) theory. In Isabelle, *locale interpretations* instantiate parametric locales by

fixing the parameters and requires proofs that they satisfy the specification of the locale. In Coq, modules can be related with subtyping relations and parametric modules can be instantiated into concrete realizations by providing arguments for the parameters [Chr03]. Other related concepts are implicit coercions and unification hints but we refer to [RK13a, Sect. 2.3&4] for a more detailed overview of framing in formal systems.

P8 Common Ground In formal systems, establishing common ground and reusing other modules is done through some form of explicit import. Direct, unnamed imports such as plain *inclusion* in Coq, *import* in Isabelle and PVS, *extension* in CASL and *includes* in MMT are standard. More complex forms of imports that allow for renamings, redefinitions or instantiation (as commonly used in CML) are supported in some systems. For instance, MMT *structures* allow importing by instantiating another theory while optionally adding new names or notations. *modules* in Coq and *imports* in Isabelle can fulfill a similar role.

In Mizar, articles can be exported as PDF files in a human readable format. The narrative documents contain a part that verbalizes the imports from the source documents and the notation reservations. This functions as a common ground section and, in fact, is similar to the informal examples for common ground from Section 2.1.

Example 3. The common ground part for [RK13b]

> The notation and terminology used in this paper have been introduced in the following papers: [4], [11], [12], [19], [9], [3], [5], [6], [21], [22], [1], [2], [7], [18], [20], [24], [25], [23], [16], [13], [14], [10], [15], and [8]. [...] In this paper T , U are non empty topological spaces, t is a point of T , and n is a natural number. (1)

Among the various systems a number of approaches are used to avoid name clashes and diamond problems including, shadowing, renaming, hiding or filtering of declarations. Again, we refer to [RK13a, Sect. 2.2&4] for a more detailed overview of imports and inheritance in formal systems.

At the foundational level, the ambiguity specific to CML is naturally not a concern in formal systems. However, the plurality of foundational assumptions and language features that are needed to effectively *do* mathematics remains important. Most formal systems take the approach of using an extremely expressive language that can be used to define various mathematical foundations internally. For instance Coq is based on the calculus of constructions (CoC [CH88]), Isabelle on higher order logic and Mizar on Tarski-Grothendieck set theory on top of a (mostly) first-order logic. MMT, on the other hand, takes a different approach by making the logical language a parameter in the representation format and allowing one to relate different logical languages via translations. We discuss MMT in more detail in Section 4.2.

P9 Structural Dimorphism The richness in constructs of formal language that we discussed above can be misleading because it studies the human-oriented surface syntax that authors write. But some of these constructs are extensions such as macros or syntactic sugar that are not present in the underlying formal language that the system processes and checks. Therefore, one can

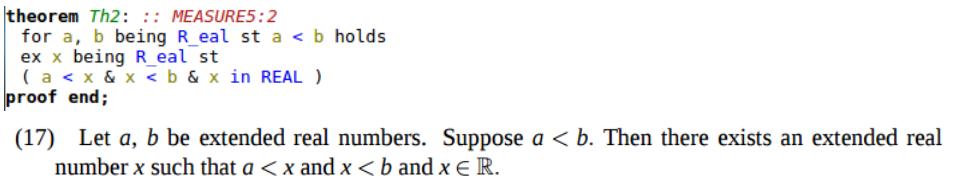


Figure 2.1: Presentation for the Mizar Mathematical Library

often distinguish between a rich *pragmatic* language level optimized for authors and a minimal *strict level* optimized for the proof system. For instance, in the Mizar case the syntax in which Mizar articles are written is called the *pattern-level* (presentation-level) language and the machine-internal representation used in the Mizar system is called the *constructor-level* (semantic level) language [IKRU13]. Isabelle achieves a similar effect by using *packages* to define user-convenient constructs on top of the core HOL primitives.

Moreover, even with the rich pragmatic level, source documents of formal systems are typically burdensome to read and understand even by habitual users. This is partly because the languages are optimized for writing and parsing rather than reading. As a consequence some libraries offer additional ways of browsing the documents by *presenting* the source documents in a more reader-friendly manner.

For instance, Mizar and Isabelle provide two ways of presenting articles, a HTML-based one and a PDF-based one. The HTML presentation is typically a source view, enhanced with syntax highlighting, hyperlinks, expandable components (i.e. sections, proofs, etc.). Figure 2.1 shows the two (HTML and PDF) presentations for Theorem 2 from the MML article `MEASURE5`.

Currently, presentation algorithms (like the ones exemplified in Figure 2.1) are relatively simple and the presentation is structurally identical to the source, different notations or renderings (e.g. LaTeX for PDF, CSS and JavaScript for HTML). However, not all presentation is generated algorithmically. For instance, [TKUG13] uses manually written LaTeX for the presentation part by cross-linking between Hales's informal Flyspeck book, and the formal Flyspeck development.

Chapter 3

State of the Art

Acknowledgement. *Part of the results in this Chapter are based on collaborative work with Michael Kohlhase that was published in [KI14]. However, the work is significantly revised and updated.*

In this Chapter we present and analyze related languages for mathematical knowledge representation and their associated systems. We observe two categories of such languages: the (mathematical) natural language processing community and the formal mathematics community.

First, there are several, traditional, target formats for common mathematical language (CML) that mix logical aspects with structural ones. For example, dynamic logics like DRT [KR93] or DPL [GS91] combines the truth functionality of logical connectives with the accessibility of discourse referents. Montague-style [Mon74] systems combine structural issues (using λ-calculi to achieve compositionality of semantics construction) again with certain – often non-standard – assumptions about the propositional level. Systems like λ-DRT [KKP96] or dynamic Montague grammar [GS90], further increase expressivity but can become complex and unwieldy. Traditionally, the most common approach towards a representation language for mathematics has been by combining all the desired traits of the various semantic formats into a single extremely expressive language that contains all other formats as sub-languages. We call this *homogeneous pluralism*) as opposed to *heterogeneous pluralism* which makes the logical language a parameter in the representation format and (optionally) relate different logical languages via translations. Here we take the heterogeneous view as motivated by the phenomena from Section 2.1. Therefore we see suitable logical representation languages (which tend to largely ignore the discourse and context levels) as parameters for such meta-level heterogeneous system rather than as direct alternatives. We discuss more concrete examples in detail below.

Second, as presented in Section 2.2, there are a wide variety of formal languages and systems for representing mathematics. They generally adopt a bottom-up approach where they fix a logic and syntax and re-develop mathematics on top of that. Therefore, representing phenomena of common mathematical language is, at best, an indirect goal. However many formal languages come with (a growing list of) narrative constructs inspired from CML. Moreover, type theories

with dependent record types (e.g. the calculus of constructions (CoC [CH88]) of the Coq system [BC04, Coq96]) can represent many of the structures at the theory level in a logic-internal way by encoding what we think of as theories as record types and views as type coersions. However, their expressivity with respect to representing CML in general and the flexiformal phenomena (from Chapter 2) in particular is still very limited. Therefore, they are not realistically practical as a basis for building human-facing applications where coverage of CML is essential because that is the language mathematicians use and understand. Moreover, since the core goal of such formal systems is verification, they require full formalization. Despite some notable achievements, such as the Flyspeck project [HAB+15] which formalized the proof of the Kepler conjecture in HOL Light [Har11] and Isabelle [Pau05] as well as the formalization of the Feit-Thompson theorem [GAA+13] in Coq, formalizing mathematics on a large scale is still prohibitively expensive. Therefore, given the particular objectives of this thesis, most formal systems are not directly pertinent. This is why we analyzed them as potential source formats in Section 2.2 rather than here as target languages.

Below, we discuss in more detail particular languages and systems that we do consider suitable targets for representing flexiformal phenomena, either because of their expressivity, design goals or ubiquity. We discuss each of the selected languages individually below and evaluate them with respect to the phenomena from Section 2.1. Figure 3.1 gives an overview of the results.

First-Order Logic On the surface, first-order logic is an appealing target format for translating mathematical documents since, together with axiomatic set theory [Ber91], it forms the most widely accepted foundation of mathematics. However, due to its first-order nature and limitations on handling some natural language phenomena, (e.g. anaphora) other variants, such as higher-order or dynamic logics [Har84] were identified as more appealing.

In general, FOL and related logics are good at representing formulae (**P1**:+), and allow for named declarations (**P1b**:+) and references (**P1c**:+). However, there is no mixing of text with formulae (**P2**:-) and there are no constructs for declaring notations (or verbalizations) (**P3**:-).

Moreover, while there are many ways to extend FOL with a more or less complex module system, modularity is not typically a part of FOL's standard formulation. Therefore, one cannot adequately represent the high level discourse (**P4**:-, **P5**:-) and context (**P6**:-, **P7**:-, **P8**:-) structure of mathematical documents (e.g. theorems, definitions, proofs, theories, etc. and the relations between them). Proofs can be naturally represented, but not their pragmatic structure with internal declarations, assumptions and proofs steps (**P5c**:0).

As a direct consequence, the phenomena from the discourse and context level discussed in Section 2 cannot be captured in plain FOL or related logics.

Moreover there is no solution as part of FOL for representing opaque (**R1**:-) or underspecified (**R3**:-) formulae. On the plus side, FOL is a relatively simple language with a well defined semantics and therefore easy to implement (**R2**:+).

	FOL	Naproche	WTT	MathLang	OMDoc	MMT	Mizar
P1 Formulae as Primitive Objects	+	+	+	+	+	+	+
P1a Symbol Application	+	+	+	+	+	+	+
P1b Naming Objects Locally	+	+	+	+	+	+	+
P1c Referential Meaning	+	+	+	+	+	+	+
P2 Mixing Text and Formulae	–	+	0	+	+	–	0
P3 Notations and Verbalizations	–	–	–	–	+	0	+
P4 Mathematical Statements	–	+	+	+	+	+	+
P4a Assertions	–	+	0	+	+	+	+
P4b Definitions	–	+	+	+	+	+	+
P4c Type Declarations	–	+	+	+	+	+	+
P5 Structured Declarations	–	–	0	0	0	0	+
P5a Inline Statements	–	–	0	+	+	+	+
P5b Mathematical Structures	–	–	–	–	0	0	+
P5c Proofs	0	+	0	+	+	0	+
P6 Modularity	–	–	0	0	0	0	0
P6a Grouping Statements	–	–	+	+	+	+	+
P6b Dynamic Theories	–	–	–	–	–	–	–
P7 Framing	–	–	–	–	+	+	–
P8 Common Ground	–	–	0	0	0	0	0
P8a Recaps	–	–	–	–	0	0	0
P8b Foundational Ambiguity	–	–	+	+	+	0	0
P9 Structural Dimorphism	–	–	–	0	0	–	–
R1 Opaqueness	–	–	0	0	+	–	–
R2 System Scalability	+	+	0	+	0	+	0
R3 Underspecification	–	–	–	–	–	–	–

Figure 3.1: Features of Mathematical Knowledge Representation Languages

Naproche The Naproche project [CFK+10] starts from the common mathematical language to develop a *controlled natural language* (CNL) for mathematical texts and a proof checking software to formally check texts written in this language. An important result is translating the first chapter of Landau's book *Grundlagen der Analysis* [Lan30] in Naproche's controlled language [CCK09] while staying relatively close to the original.

The Naproche CNL focuses on proofs to permit adequate proof checking by the Naproche system and does not necessarily aim to cover all phenomena from Section 2.1.

At the phrase level, the Naproche CNL includes a language for formulae which can be integrated within natural language phrases (**P1**:+, **P1a**:+) and includes local declarations (**P1b**:+) and references (**P1c**:+). Moreover, as its goal is to remain as close as possible to CML, it supports mixing text and formulae, albeit in a controlled way (**P2**:+). However, there is no explicit support for declaring notations and verbalizations (**P3**:-).

At the discourse level, Naproche's CNL offers explicit markup for *axioms*, *definitions*, *lemmas* and *theorems* (**P4**:+) as well as *proofs* with *case distinctions* and *assumptions* (**P5c**:+) but not for inline statements or mathmatical structures (**P5a**:-, **P5b**:-). Proofs are represented using *proof representation structures* (PRS) which are adapted from discourse representation structures [KR93].

The context level, as defined in Section 2.1, is missing as there is no explicit markup for documents, theories or groups of statements, let alone relations between them (**P6**:-, **P7**:-, **P8**:-).

The Naproche project also includes a system implementing the language (**R2**:+). Some ambiguity is supported in the syntax but it is meant to be resolved during parsing and analysis. Therefore, the system does not allow representing opaque (**R1**:-) or underspecified (**R3**:-) content.

Weak Type Theory WTT (Weak Type Theory) [KN04] is a refinement of de Bruijn's Mathematical Vernacular and is designed to act as an intermediary between common mathematical language and formal mathematics based on various logics.

At the phrase level, WTT has primitives inspired by common mathematical language for constructing terms (**P1**:+) such as application (**P1a**:+) , binding (**P1b**:+) and referencing (**P1c**:+). The structure of narrative text interspersed with formulae is partly captured with dedicated primitives such as *nouns*, *adjectives* and *sets*. But the distinction between narrative text and formulae is lost as they are both represented within the same WTT concrete syntax (**P2**:0). Moreover, WTT does not support declaring notations or verbalizations (**P3**:-).

At the statement level, WTT has *statements* and *definitions* (**P4**:+, **P4b**:+) although it does not distinguish, for instance, examples or lemmas (**P4a**:0). The approach to representing types is heavily inspired by the CML practice and uses *weak types* constructed by predicates over sets (or linguistically, adjectives over nouns). The representation of mathematical types is, therefore, one of the main strengths of Weak Type Theory **P4c**:+.

Structured statements and proofs can be represented albeit slightly awkwardly. For this, WTT uses *contexts* which are ordered and can therefore be linearized (**P5a**:0). Moreover, contexts

can contain assumptions for proofs which are represented as statements inside contexts (**P5c**:0). However, mathematical structures additionally require a form of extension and reusability which is not provided (**P5b**:-).

Books are the only module level element in WTT, roughly representing ordered collections of statements (**P6**:0, **P6a**:+, **P6b**:-). Therefore, the context level of WTT is limited in expressivity, for instance with respect to framing (**P7**:-) or complex knowledge sharing via parametric imports (**P8a**:-). However, it does not require committing to a specific logic or foundation (**P8b**:+, **P8**:0).

As WTT is designed to be structurally close to the grammar of natural language it is suitable as a first step in the formalization process. Consequently, translating a CML text into WTT is easier than fully formalizing it (although further formalization can require significant effort) [Joj05, Gel04]. This is because, while WTT requires *linguistic correctness* and adherence to the WTT grammar, it does not require providing a precise semantics (**R1**:0). While there is no full implementation of WTT [KN04] we know of, the formal syntax makes it machine-friendly (**R2**:0). With respect to underspecification, WTT formalization requires solving ambiguities to the point where the WTT grammar can resolve them (**R3**:-).

MathLang MathLang [KW08] aims at reaching a compromise between expressivity and formality and thus provide an interface language between mathematicians and computers as well as a framework to make the link with existing formal proof systems. MathLang is based on WTT and can be viewed as an extension of it. Therefore, we evaluate it relative the coverage of WTT which we discussed above.

The current design of the MathLang language is composed of three *aspects* [KW08].

- The Core Grammar Aspect (CGa) is a kind of weak type system directly based on WTT [KN04] and de Bruijn's mathematical vernacular [dB87]. In addition to WTT, MathLang introduces *flags* as nestable context-management construct to better represent the scope of variables and assumptions over several lines. More, MathLang adds *blocks* as a nestable construct for representing the narrative structures such as proofs, examples, sections or paragraphs. Blocks also allow defining local constants which is useful, for instance, in representing local lemmas during proofs. MathLang *references* allow statements and definitions to refer to previous lines. As a result, compared to WTT, support for both plain and structured statements is significantly improved (**P4a**:+, **P5a**:+, **P5c**:+). However, due to the lack of OOP-style instantiation or extension, support of representing mathematical structures is limited (**P5b**:-).

- The Text and Symbol Aspect (TSa) enables establishing the association between textual presentation and mathematical meaning. A notion of *souring* rules (named in relation to syntactic *sugaring*) is introduced to permit better control over the structure of mathematical texts. Although mostly annotation based and at the object level, the effort to weave together presentation-oriented and content-oriented representations tackles structural dimorphism (**P9**:0).

- The Document Rhetorical Aspect (DRa) allows labeling fragments of text and establishing relations between them using unary (labels) or binary (relations) predicates. For instance a text fragment can be labeled as a *chapter*, or *section* or *theorem*. Then a *proof* fragment can be related to the theorem it proves.

Just like WTT, MathLang lacks at the module level though where it only has *books* as ordered collections of statements (**P6**:0) Even though DRa aspect improves representation of narrative structures in documents, it offers no real solution for framing (**P7**:-) or common ground (**P8**:-).

However, Unlike WTT, MathLang also comes with a system [KMW04] that implements the abstract syntax, type checking (in the sense of MathLang types) as well as a presentation functionality via LaTeX (**R2**:+).

OMDoc OMDoc [Koh06, Koh10] is a rich representation language for mathematical knowledge with a large set of primitives motivated by expressivity and user familiarity. It models mathematical knowledge at three levels: *i*) *object level* for formulae, expressions or equations *ii*) *statement level* for symbol declarations, definitions, axioms, theorems or notation definitions and *iii*) *module level* for theories and views. Additionally, it adds an infrastructure for representing the functional aspects of mathematical documents at the content markup level. Note that we discuss OMDoc in more detail in Section 4.1 and only give an overview here.

OMDoc has been successfully used as a representational basis in a wide array of applications ranging from theorem prover interfaces to e-learning systems. To allow this diversity of applications, the format has acquired a large, interconnected set of language constructs. This was motivated by coverage and user familiarity but not by minimality and orthogonality of language primitives. As a result the OMDoc language is expressive enough to provide good coverage for most of the phenomena above.

At the phrase level, it mixes informal text (HTML) with dedicated formula markup in either OpenMath or MathML (**P1**:+, **P1b**:+, **P1c**:+, **P3**:+). Moreover, it allows mixing narrative text and formulae (**P2**:+).

At the discourse level OMDoc provides a rich set primitives including *definitions*, *assertions*, *example* and *exercise* (**P4**:+, **P4a**:+, **P4b**:+, **P4c**:+). OMDoc proofs are also particularly expressive [ASC06] (**P5c**:+). With respect to structured declarations, OMDoc provides primitives for inline statements but not for mathematical structures (**P5**:0, **P5a**:+, **P5b**:0).

At the context level, OMDoc omdoc provides *theories* to organize declarations (**P6a**:+). However, OMDoc theories are fixed unlike mathematical ones (**P6b**:-, **P6**:0). OMDoc also provides *views* to relate theories (**P7**:+) as well as *imports* and *structures* to reuse knowledge (**P8a**:0). Also, OMDoc permits but does not require committing to a particular logic or foundation (**P8b**:+, **P8**:0).

At all levels, OMDoc allows mixing informal text (or HTML) within or alongside OMDoc elements (**R1**:+). Moreover, OMDoc provides dedicated elements for partially representing structural dimorphism. At the object level it supports MathML. At the statement level it provides

a CMP element for commented mathematical properties and a FMP element formal mathematical properties. But it is lacking in this sense at the theory and document level (**P9**:0). Also, OMDOC's focus on coverage and expressivity has meant that it lacks a fully specified semantics. Therefore, the only implementation [JOM] (now obsolete) is limited in functionality and partial in coverage (**R2**:0). There is also no solution for representing underspecification (**R3**:-).

MMT MMT [RK13a] is a generic, formal module system for mathematical knowledge and is a basis for foundation-independent knowledge representation. Relative to OMDOC, MMT is a complete redesign of the formal core of OMDOC focusing on foundation-independence, scalability, modularity and minimality. As for OMDOC, we discuss MMT in more detail in Section 4.2 and only give an overview here.

At the object level, MMT uses OPENMATH-inspired primitives for application, binding and referencing symbols or variables (**P1**:+, **P1a**:+, **P1b**:+, **P1c**:+). However, it does not allow mixing text with formulae (**P2**:+). MMT optionally has a notation [IR12b] associated with each symbol but its notation language is limited. Additionally it does not distinguish between notations and verbalizations and has limited support for notation variants (e.g. "C_n^k" and "$\binom{n}{k}$" for combinations) (**P3**:0).

At the statement level, MMT uses constant declarations as a single yet generic mechanism for representing a wide array of mathematical statements (**P4**:+). There is no native appropriate construct for representing structured declarations but *declaration patterns* are an extension introduced in [HKR12] that can be used to represent inline declarations (**P5a**:+). Proofs are represented as MMT terms, with no internal structure (**P5c**:0). For mathematical structures, MMT theories can be used to represent their inner structure, albeit at the theory instead of the statement level (**P5b**:0, **P5**:0).

At the theory level, MMT uses theories for organizing declarations (**P6a**:+) but there is no construct for representing dynamic theories (**P6b**:-, **P6**:0). MMT views formalize the mathematical intuition of framing (**P7**:+) and MMT includes and structures are used to reuse knowledge and establish common ground (**P8a**:0). MMT is logic and foundation independent but does require individual theories to fix a logic (**P8b**:0, **P8**:0).

As a fully formal language MMT is designed to represent only the content aspect of mathematical knowledge. Therefore support for structural dimorphism, opaqueness and underspecification is lacking (**P9**:-, **R1**:-, **R3**:-). However, MMT does come with an associated, scalable implementation which we described in Section 4.2 below (**R2**:+).

Mizar MIZAR [TB85] is a representation language for formal mathematics but designed to be as close as possible to the mathematical vernacular. The MIZAR language is based on Tarski-Grothendieck set theory [Try90] formalized in (unsorted) first-order logic[1](**P1**:+, **P1a**:+, **P1b**:+, **P1c**:+).

[1]Technically, theorem schemes and the Fraenkel operator of MIZAR transcend first-order expressiveness, but the language is first-order in style.

MIZAR notations are effectively mixfix, text-based notations. Moreover, while not explicitly having verbalizations the names of dependent or refined types are composed to produce adequate verbalizations such as `empty` **`set`** or `finite Subset` **`of`** `NAT` (**P3**:+). Therefore, even though, as a formal language it does not allow arbitrary text within formulae, the syntax simulates it somewhat (**P2**:0).

At the statement level, MIZAR has number of primitives to represent mathematical statements, including *justified theorems*, *function* and *predicate definitions* and *schemes* (which are second order sentences which take functors and predicates as arguments) (**P4**:+, **P4a**:+,(**P4b**:+)). Additionally, MIZAR provides an expressive and flexible type system that features dependent types as well as predicate restrictions [Ban03]. The central primitives are *mode* and *attribute definitions* as well as *clusters* all of which are used to define new types and refine or compose existing ones (**P4c**:+). MIZAR *structure definitions* allow for defining pragmatic concepts with internal structure and give *selectors* for accessing its internals (pragmatics) (**P5**:+, **P5a**:+, **P5b**:+). Additionally, MIZAR *proofs* can also have complex internal structure including reasoning steps, local assumptions, proofs by cases, etc. (**P5c**:+).

At the document context level, Mizar organizes declarations in *articles* (**P6a**:+) that are fixed and defined declaratively by the statements they contain (**P6b**:-, **P6**:0). There is no support for framing (**P7**:-) and common ground is established only through direct, unnamed imports (**P8**:0, **P8a**:0). Mizar uses Tarksi-Grothendieck set theory (TG) [Try90] as a fixed foundation. However TG, is a relatively expressive one that can represent others (e.g. ZFC) internally **P8b**:0).

Since MIZAR is a fully formal language there is also no support for structural dimorphism (**P9**:-), opaqueness (**R1**:-), or underspecification (**R3**:-). However, there is an implementation (the Mizar System [Miz]), although the system is notoriously complex and abstruse (**R2**:0).

Part II

Representing Flexiformal Knowledge

Chapter 4

Preliminaries

As observed by studying related work in Chapter 3 none of the languages we looked at cover all the phenomena we observed in practice and presented in Chapter 2. Moreover, building a language and system for representing mathematics that even partially covers our phenomena is a multiple person-decade endeavor.

Therefore, our approach in this thesis is to develop an experimental language and system by extending already existing ones. Concretely we base our design on the OMDoc [Koh06, Koh10] and Mmt [RK13a] languages.

OMDoc is one of the more expressive languages surveyed and has the most extensive coverage of narrative phenomena. However, due to its complexity, it does not have a formal semantics and does not lend itself to a scalable implementation.

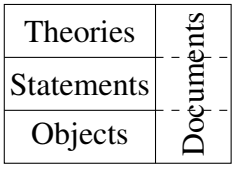
Mmt is interesting due to its extensibility at both the language and system level as well as its coverage of essential phenomena. Specifically, the module-level appears to be the least understood and covered by most mathematical representation languages. We believe that Mmt's theory graph-based solution is the correct approach and the most mature realization. Moreover, Mmt is parametric in the choice of foundation as it allows arbitrary declarative languages such as logics or type theories to be represented in it. For the scope of this thesis, this allows us to abstract over foundational details and focus on the meta-level constructs when tackling the flexiformal phenomena from Chapter 2. However, Mmt is fully formal and lacks any real coverage of opaqueness or narrative structures.

In Chapter 5 and later we will extend the Mmt language to cover missing phenomena inspired mostly by the OMDoc language. Before that, below, we briefly review the parts of OMDoc and, respectively, Mmt that are relevant in this thesis.

4.1 The OMDoc Format

OMDoc [Koh06, Koh10], as discussed in Section 2.1, is not a direct solution to the problem

tackled here. However, it is one of the most expressive languages we reviewed in terms of coverage of the phenomena. Moreover, OMDOC is one of the only languages that recognizes the narrative-content duality as an intrinsic property of informal mathematics and comes with dedicated primitives to adequately represent it.



For instance, at the object level, OMDOC can use MATHML which in turn uses *parallel markup* to represent these two aspects. Parallel markup is realized in MATHML using the `semantics` and `annotation-xml` elements. The former has the narration (or *presentation*) markup as the first child followed by optional `annotation-xml` elements which contain the content markup. Corresponding subtrees are marked with cross-references using the `id` and `xref` attributes. Consider the example in the listing below which represents the Boolean expression $a(b + c)$ (inspired from an example in the MATHML 3.0 W3C Recommendation in [ABC+10]). The first child is the presentation markup with an `id` attribute on each element. The `mrow`, `mi`, and `mo` elements are part of the MATHML presentation markup primitives, representing rows, identifiers and operators, respectively. The second child annotates the presentation with its semantics, where each element has a `xref` attribute referring to the corresponding presentation element. The `apply`, `ci`, `and`, and `or` elements are part of the MATHML content markup primitives for representing applications, identifiers, conjunctions and disjunctions, respectively.

```
<semantics>
  <mrow id="E">
    <mrow id="E.1">
      <mi id="E.1.1">a</mi>
    </mrow>
    <mo id="E.2">&#x2062;</mo> <!--INVISIBLE TIMES-->
    <mrow id="E.3">
      <mo id="E.3.1">(</mo>
      <mi id="E.3.2">b</mi>
      <mo id="E.3.3">+</mo>
      <mi id="E.3.4">c</mi>
      <mo id="E.3.5">)</mo>
    </mrow>
  </mrow>
  <annotation-xml encoding="MathML-Content">
    <apply xref="E">
      <and xref="E.2"/>
      <ci xref="E.1.1">a</ci>
      <apply xref="E.3">
        <or xref="E.3.3"/>
        <ci xref="E.3.2">c</ci>
        <ci xref="E.3.4">d</ci>
      </apply>
    </apply>
  </annotation-xml>
</semantics>
```

Note that the same presentation $a(b+c)$ could have a different content interpretation. For instance

if $+$ and invisible times are sum and product between real numbers instead of boolean operators. Moreover, the same content could have a different presentation such as $a \wedge (b \vee c)$. Therefore, both the presentation and content representations are needed together to adequately capture the mathematical expression in question.

At the statement level, OMDOC identifies a number of conceptual primitives and provides its own infrastructure for marking them up. In particular, it supplies dedicated elements for representing *axioms*, *definitions*, *assertions* (theorems, conjectures, lemmas, etc.) as well as proofs.

OMDOC accounts for the *dynamic vocabulary* of mathematics using the `symbol` and `notation` elements which allow for declaring new symbols and, respectively, notations for them. Symbol declarations introduce a new name, and notation elements can use that name to declare a notation for its associated symbol. `notation` elements have two children: 1. the `prototype` element encodes the functional structure using content MATHML with additional `expr` elements for the parameters, 2. the `rendering` element encodes the presentation using presentation MATHML with additional `render` elements which represent the (recursive) presentation of parameters.

Consider the example in the listing below, which represents the notation $SL(n, F)$ for the special linear group of degree n over a field F (see Example 2.12 from 2.1). The listing declares a new symbol for the special linear group then a notation for it. The prototype of the notation encodes the application of the symbol to two arguments which in the example rendering above they would be n and F. Then the rendering encodes the presentation with the `render` elements as placeholders for the presentation of each argument. Note that we use the `m` namespace for native MATHML elements (as opposed to OMDOC ones) for clarity.

```
<symbol name="special-linear-group"/>
<notation for="special-linear-group">
  <prototype>
    <m:apply>
      <m:csymbol>special-linear-group</m:csymbol>
      <expr name="arg1"/>
      <expr name="arg2"/>
    <m:apply>
  </prototype>
  <rendering>
    <m:mrow>
      <m:mo>SL</m:mo>
      <m:mo>(</m:mo>
      <render name="arg1">
      <m:mo>,</m:mo>
      <render name="arg2">
      <m:mo>)</m:mo>
    </m:mrow>
  </rendering>
</notation>
```

OMDOC also accounts for the narrative-content duality through a form of parallel markup at the statement level. Concretely, each statement-level element has two children elements `CMP` (commented mathematical properties) and `FMP` (formal mathematical properties) for representing the

narrative and, respectively, content aspect. The internal formats used for each are parametric but are typically HTML and presentation MATHML inside the CMP and OPENMATH or content MATHML inside the FMP. Consider, for instance, the example in the listing below which represents the OMDOC markup for Fermat's little theorem: "If p is a prime number and a is a natural number, then $a^p \equiv a \pmod{p}$." (see Example 2.14 from Section 2.1). The CMP below effectively encodes the statement above in HTML plus MATHML. Then, the FMP uses various content MATHML primitives to represent its formal semantics. Effectively, it encodes the following formal statement:

$$\texttt{prime-number}(p) \wedge \texttt{natural-number}(a) \Rightarrow \texttt{rem}(a^p, p) = \texttt{rem}(a, p)$$

where $\texttt{rem}(m, n)$ represents the remainder of m when divided with n.

```
<assertion type="theorem" name="Fermat's Little Theorem">
  <CMP>
    <p>
      If <math><mi>p</mi></math> is a prime number and <math><mi>a</mi></
         math> is a natural number, then
      <math>
        <mrow>
          <msup>
            <mi>a</mi>
            <mi>p</mi>
          </msup>
          <mo>≡</mo>
          <mi>a</mi>
          <mo>(</mo> <mo>mod</mo> <mi>p</mi> <mo>)</mo>
        <mrow>
      </math>
    <p>
  </CMP>
  <FMP>
    <apply>
      <implies/>
      <apply>
        <and/>
        <apply> <csymbol>prime-number</csymbol> <ci>p</ci> </apply>
        <apply> <csymbol>natural-number</csymbol> <ci>a</ci> </apply>
      </apply>
      <apply>
        <equal/>
        <apply>
          <rem/>
          <apply> <power/> <ci>a</ci> <ci>p</ci> </apply>
          <ci>p</ci>
        </apply>
        <apply> <rem/> <ci>a</ci> <ci>p</ci> </apply>
      </apply>
    </apply>
  </FMP>
</assertion>
```

At the module level OMDOC also provides its own conceptual primitives such as theories, imports and views. However, in re-developing the formal core of OMDOC, MMT reuses the same module-level primitives but with a more thorough specification and a formal semantics. Therefore, we discuss these as part of introducing MMT in Section 4.2 below.

At the document level OMDOC provides elements for metadata (using the `metadata` element), front- and back-matter such as `tableofcontents`, `bibliography` and `index`, as well as top-level elements such as theories or views. Moreover, the `omgroup` element is used for generic grouping and can be used to instantiate sectioning, sequences of paragraphs, itemizes or enumerations.

Other than the native XML language, the primary surface syntax for writing OMDOC is the LaTeX-based sTeX [Koh08, sTe] language. We discuss sTeX and OMDOC further in Section 7.2 where we import several sTeX libraries into our extension of MMT to asses its coverage with respect to OMDOC.

4.2 The MMT Language and System

4.2.1 The MMT Language

The MMT language [RK13a] is designed to be applicable to a large collection of declarative formal base languages and all MMT notions are fully abstract in the choice of the base language. Therefore, MMT focuses on foundation-independence, scalability and modularity.

It is meant to be applicable to all base languages based on *theories*. Relations between theories are represented as MMT *views* (or *theory morphisms*). Theories and views form the MMT *module* level. A *view* $V : T_1 \to T_2$ is based on a *signature morphism* $V^\sigma : T_1 \to T_2$ interpreting all symbols of T_1 as terms in T_2. But, additionally, V translates all theorems of T_1 to theorems of T_2. MMT uses the *Curry-Howard representation* to drop the distinction between symbols and axioms (and thus between signatures and theories). As a result, MMT needs only theories and views.

We will only give a brief overview of MMT and refer to [RK13a] for details. Furthermore, we will omit some details for simplicity, most notably MMT structures and documents. However, both will be recovered as extensions in Section 6. We give the grammar of the relevant MMT fragment in Figure 4.2 and discuss each production below (see also the MMT ontology in Figure 4.1).

Remark 4. We will use the following notations throughout this thesis. To represent instances of concepts, we will use the corresponding identifiers from the grammar productions of the concept (see Figure 4.2). However, when needed, we use subscripts to distinguish them. For example, since T is the identifier for theories, we will routinely use T_1 and T_2 as theory names. Moreover, we use superscripts as labels to clarify the meaning of certain instances. For example, a body (Σ) which we know contains exclusively constants (C) might be denoted by Σ^C or possibly Σ_1^C. In particular, we will often use g^M for a module URI and g^S for a statement URI.

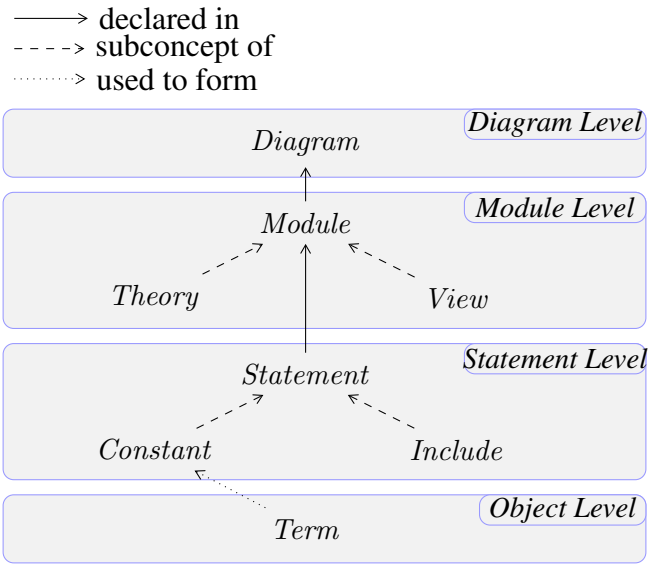


Figure 4.1: The MMT Ontology

Declaration	D	::=	$M \mid S$
Diagram	Θ	::=	M^*
Body	Σ	::=	S^*
Module Level			
Module	M	::=	$T \mid V$
Theory	T	::=	$g = \{\Sigma\}$
View	V	::=	$g : g \to g = \{\Sigma\}$
Statement Level			
Statement	S	::=	$C \mid I$
Constant	C	::=	$l[:E][=E]$
Include	I	::=	`include` g
Object Level			
Object	O	::=	$E \mid \Gamma$
Expression	E	::=	$l \mid g \mid E\ E^* \mid E\ \Gamma.E$
Context	Γ	::=	$[(l[:E][=E])^*]$
Identifiers			
Global name	g	::=	$URI[?l[?l]]$
Local name	l	::=	`String`

Figure 4.2: MMT Grammar

The central notion is that of a *diagram*[1] (or *theory graph*), a list of modules, which are theories T or views V. Correspondingly we use the term *body* to refer to a list of statements, which are includes I or constants C. Moreover, we use the term *declarations* to collectively refer to both modules and statements.

A **theory** declaration $g = \Sigma$ introduces a theory with global name g and body Σ containing a list of statements.

View declarations (or theory morphisms) $g : g_1^T \to g_2^T = \Sigma$ introduce a morphism with global name g from g_1^T to g_2^T and body Σ containing a list of statements. Such a morphism must contain exactly one constant $c := E_2$ for every undefined constant $c : E_1$ in g_1^T where E_2 is a valid term over g_2^T. Theory morphisms extend homomorphically to a mapping of g_1^T-terms to g_2^T-terms.

Intuitively, a theory morphism formalizes a translation between two formal languages. For example, the inclusion from the theory of semigroups to the theory of monoids (which extends the former with two declarations for the unit element and the neutrality axiom) can be formalized as a theory morphism. Similarly, assuming the corresponding theories are defined, the notion that the integers form a group can be formalized in MMT as a view from the theory *group* to the theory *integers*.

A **constant** declaration $l : E_1 = E_2$ introduces a constant named l with **type** E_1 and **definiens** E_2. An include statement `include` g in a module M makes all the statements from g (or accessible in g) accessible in M. We discuss *accessibility* (or *visibility*) in more detail in Section 4.2.2 below. Note that the local name of an include statement `include` g is produced by wrapping (and escaping) the global name of the included module, written $\lfloor g \rfloor$. Again, we refer to [RK13a] for details.

Terms E over a theory g^T are formed from either references g^S to constants visible from g^T, references l to bound variables, applications $E\,E_1\,\ldots\,E_n$ of a function E to a sequence of arguments, bindings $E_1\Gamma.E_2$ using a binder E_1, a bound variable context Γ, and a scope E_2. MMT terms are fragment of the OPENMATH language [BCC+04].

Every MMT module or constant declaration is identified by a canonical, globally unique URI. For modules this is their global name g, for constants $l : E_1 = E_2$ in a module g it is $g?l$. MMT statements subsume most semantically relevant statements in declarative mathematical languages including function and predicate symbols, type and universe symbols, and — using the Curry-Howard correspondence — axioms, theorems, and inference rules. Their syntax and semantics is determined by the foundation, in which MMT is parametric.

Content in the MMT language can be serialized into XML similar to OMDOC. We omit the details here but, to give the general idea, we show a simple example in the listing below. It contains the XML serialization for a trivial MMT theory `Nat-Ext` that includes the basic theory of natural numbers and uses its constants to define *one* as the *successor* of *zero*. Note that, for objects, MMT uses the same XML tags as OPENMATH, namely `OMA` for application, `OMS` for constant (symbol) references, `OMV` for variable references and `OMBIND` for binding. Moreover,

[1] The intuition is that theories are nodes and views are arrows between them.

Judgment	Intuition
$\vdash \Theta$	Θ is a well-formed diagram
$\Theta \vdash^{+} M$	adding M at the end of Θ preserve well-formedness of Θ
$\Theta \vdash^{+}_{g^M} S$	if g^M is the URI of a module M in Θ, then adding S at the end of the body of M preserves well-formedness of Θ
$\Theta, \Gamma \vdash_{g^M} E$	expression E is well-formed over theory g^M and context Γ in diagram Θ
$\vdash \Sigma(l) \Rightarrow S$	Σ contains statement S with name l (we use $\lightning$ if Σ contains no statement at l)
$\vdash \Theta(g) \Rightarrow D$	looking up g in Θ yields declaration D (or $\lightning$ for no declaration at g in Θ)
$\Theta \vdash^{v}_{g^M} g^S$	the statement at global name g^S is visible from module at g^M in Θ

Figure 4.3: MMT Judgments

for constant references, it uses the attribute name `cd` (from *content dictionary*) to give the MMT theory.

```
<theory name="Nat-Ext">
  <include from="Nat"/>
  <constant name="one">
    <type>
      <OMS cd="Nat" name="nat"/>
    </type>
    <definiens>
      <OMA>
        <OMS cd="Nat" name="succ"/>
        <OMS cd="Nat" name="zero"/>
      </OMA>
    </definiens>
  </constant>
</theory>
```

Remark 5. Note that, although we introduced local names l as plain strings, in the rest of the thesis we will occasionally use the "/" separator within local names for artificial scoping. Alternatively we could have defined local names as a non-empty list of "/"-separated strings (which is how they are actually implemented), but opted for simplicity so that the grammar is easier to understand.

4.2.2 Inference System

The judgments needed to define well-formedness are given in Figure 4.3 together with their intuitive semantics.

Remark 6. For brevity and by abuse of notation, we will occasionally directly write $\vdash M(l) \Rightarrow S$ as a judgment. This represents that *the body* of the module M contains statements S at local name l.

The rules in Figure 4.4 govern the building of diagrams by adding statements one by one. For

simplicity, all rules omit the hypotheses necessary for well-*typed*ness. For example, $\Theta \vdash^+_g l : E_1 = E_2$ requires that E_2 has type E_1.

The rules in Figure 4.5 define the well-formed expressions relative to a diagram Θ and a module in Θ identified by g^M. We omit the straightforward but tedious rules for binders as well as the ones for views (and refer to [RK13a]).

Finally, we give the rules visibility (accessibility) rules for MMT theories in Figure 4.6. They determine when a statement can be referenced which is used to define the well formedness of MMT terms (see rule E_{const_ref}). The intuition of rule vis_base is that all constants from the current theory are visible. Then, rule vis_incl states that all statements that are visible from an included theory are also visible from the current theory (i.e. inclusion is transitive).

$$\frac{}{\vdash \cdot}\,\Theta_{base} \qquad \frac{\vdash \Theta(g^M) \Rightarrow \lightning}{\Theta \vdash^+ g^M = \{\cdot\}}\,\Theta_{add\_thy}$$

$$\frac{\vdash \Theta(g^M) \Rightarrow \lightning \quad \vdash \Theta(g^T_1) \Rightarrow T_1 \quad \vdash \Theta(g^T_2) \Rightarrow T_2}{\Theta \vdash^+ g^M : g^T_1 \to g^T_2 = \{\cdot\}}\,\Theta_{add\_view}$$

$$\frac{\vdash \Theta(g) \Rightarrow T \quad \vdash \Theta(g?l) \Rightarrow \lightning \quad [\Theta, \cdot \vdash_g E_1] \quad [\Theta, \cdot \vdash_g E_2]}{\Theta \vdash^+_g l[: E_1][= E_2]}\,T_{add\_cons}$$

$$\frac{\vdash \Theta(g_1) \Rightarrow M_1 \quad \vdash \Theta(g_2) \Rightarrow M_2}{\Theta \vdash^+_{g_1} \texttt{include}\ g_2}\,T_{add\_incl}$$

Figure 4.4: Basic Rules for Building Diagrams

4.2.3 The MMT System

The MMT System [RK13a, Rab14a] (MMT API) is a Scala-based [Sca] open source implementation of the MMT language as described above. It implements basic services generically such as retrieval of content from local storage (via the MMT *backend*) as well as directly accessing content via the MMT web server using a RESTful API. Moreover, it implements a query engine based on the QMT [Rab12] query language which permits applications to use the MMT system as a semantic database. Other core services include change management [IR12a] and notation-based parsing [IR12b].

Like the MMT language, the MMT system is implemented foundation-independently and all provided services carry over to the individual languages that are represented in MMT. However,

$$\frac{\Theta \vdash^{v}_{g_M} g^S}{\Theta, \cdot \vdash_{g^M} g^S} E_{const\_ref} \qquad \frac{}{\Theta, \cdot \vdash_{g^M} [\cdot]} \Gamma_{base}$$

$$\frac{\Theta, \cdot \vdash_{g^M} \Gamma \quad [\Theta, \Gamma \vdash_{g^M} E_1] \quad [\Theta, \Gamma \vdash_{g^M} E_2]}{\Theta, \cdot \vdash_{g^M} \Gamma, x[:E_1][=E_2]} \Gamma_{add}$$

$$\frac{\vdash \Theta(g^M) \Rightarrow M \quad \Theta, \cdot \vdash_{g^M} \Gamma \quad \vdash \Gamma(x) \Rightarrow x[:E_1][=E_2]}{\Theta, \Gamma \vdash_{g^M} x} E_{var\_ref}$$

$$\frac{\Theta, \Gamma \vdash_{g^M} E_i \quad i = 0, \ldots, n}{\Theta, \Gamma \vdash_{g^M} E_0\, E_1 \ldots E_n} E_{apply} \qquad \frac{\text{accordingly}}{\Theta, \Gamma \vdash_{g^M} E_1\, \Gamma_1.E_2} E_{bind}$$

Figure 4.5: Rules for Building Expressions

$$\frac{\vdash \Theta(g) \Rightarrow g = \{\Sigma\} \quad \vdash \Sigma(l) \Rightarrow l[:E_1][=E_2]}{\Theta \vdash^{v}_{g} g?l} vis\_base$$

$$\frac{\vdash \Theta(g_1) \Rightarrow g_1 = \{\Sigma\} \quad \vdash \Sigma(\lfloor g_2 \rfloor) \Rightarrow \texttt{include}\ g_2 \quad \Theta \vdash^{v}_{g_2} g^S}{\Theta \vdash^{v}_{g_1} g^S} vis\_incl$$

Figure 4.6: Visibility Rules for MMT Theories

some services cannot be implemented generically, and therefore, the MMT API provides a rich set of abstractions that permit extending the core functionality of the MMT system using *plugins*.

The essential plugins that are most relevant for this thesis are as follows:

- *Importers* are MMT plugins that parse and process input source files from various concrete languages and produce MMT abstract syntax. They allow MMT to support multiple *surface syntaxes* so users can use their own language and syntax if they implement an MMT importer for it.
- *Exporters* are MMT plugins that are the opposite of importers in the sense that they produce output files based on MMT content. The typical example is presenting MMT modules for instance as HTML documents [KMR08].
- *Servlets* (or *server plugins*) are MMT plugins that extend the functionality of the MMT web server. Servlets allow external application designers (most commonly web applications) to

implement their own interface to the core MMT API as needed by their application.

- *Frameworks* (or *foundations*) are MMT plugins that "give the semantics" for concrete languages represented in MMT by implementing oracles to answer whether typing or equality constraints hold in that language. They allow MMT to implement other algorithms, such as type checking, generically and defer to foundation plugins when necessary.

Additionally, in the implementation, MMT allows annotating each declaration with arbitrary information as meta-data using the `meta` tag. This information is normally ignored by the core MMT processes but MMT plugins can access and make use of it. A typical example is annotating constants with information about how they should be presented to be used by selected exporters.

The extensibility of the MMT system is one of its central features as it allows it to scale to a wide range of potential applications. Moreover, it is critical because the generality of the MMT language means that many basic services cannot be fully implemented at the MMT level (e.g. type checking needs input from framework plugins). As we will see in future Chapters this extensible design is essential for balancing two conflicting requirements: being generic enough to support a wide array of heterogeneous languages while remaining useful for enabling development of practical applications.

Chapter 5

A Formal Language for Flexiformal Knowledge

In this Chapter, we first look at the shortcomings of MMT with respect to the phenomena from Chapter 2 and design language extensions to tackle them. We present them below in Section 5.1. Then, we integrate the extensions into the IMMT language and system, which we discuss in Section 5.2. Later, in Chapter 6, we evaluate the resulting language by applying it to represent the concrete phenomena discussed above.

5.1 MMT Extensions for Flexiformal Knowledge

Acknowledgement. *Part of the results in this Section, in particular in Section 5.1.3 and Section 5.1.4, are based on collaborative work with Michael Kohlhase and Florian Rabe and have been prepared for publishing in [IKR].*

As discussed in Chapter 3, MMT already meets a number of flexiformal requirements. We consolidate the deficiencies into two central shortcomings of the MMT design with respect to our phenomena. Firstly, MMT is (by design) fully formal so it cannot adequately represent content that is either fully or partially informal. Secondly, MMT has a minimal, fixed set of declaration-level constructs that are not sufficient to directly represent (in a structure-preserving way) the breadth of advanced structuring constructs used in both formal and informal knowledge representation languages.

Therefore, we extend both the object and declaration level of MMT to tackle the shortcomings mentioned above by adding:

- *opaque terms* to represent partially formalized content (**R1**)
- *bipartite terms* to capture structural dimorphism at the object level (**P9**) as well as the mixing of narration and formulae in CML texts (**P2**)

Phenomenon	Realization	Discussed in
P1 Formulae as Primitive Objects	MMT terms	[Rab14b]
P2 Mixing Text and Formulae	Bipartite Objects	5.1.2
P3 Notations and Verbalizations	Bipartite Objects and Structural Features	6.1.1
P4 Mathematical Statements	MMT Constants + MetaData	[RK13a]
P5 Structured Declarations	Structural Features	5.1.4
P5a Inline Statements	MMT + Patterns	[HKR12]
P5b Mathematical Structures	$\mathfrak{MathStruct}$ Feature	6.1.2
P5c Proofs	$\mathfrak{Proof}$ Feature	6.1.3
P6 Modularity	MMT + Structural Features	[RK13a],5.1.4
P6a Grouping Statements	MMT Theories	[RK13a]
P6b Dynamic Theories	$\mathfrak{Realm}$ Feature	[CFK14],6.1.4
P7 Framing	MMT Views	[RK13a, Rab15b]
P8 Common Ground	MMT + Structural Features	5.1.4
P8a Recaps	MMT Imports + $\mathfrak{Realm}$ Feature	6.1.5
P8b Foundational Ambiguity	MMT + $\mathfrak{PreRealm}$ feature	6.1.6
P9 Structural Dimorphism	Bipartite Declarations	5.1.3
R1 Opaqueness	Opaque Terms	5.1.1
R2 System Scalability	MMT System + IMMT Extensions	[Rab13], 5.2.3
R3 Underspecification	MMT System (partial)	[Rab13]

Figure 5.1: Phenomena and their Realization in IMMT

- *bipartite declarations* to cover structural dimorphism at the statement and module levels (**P9**)
- a mechanism for declaring *structural features* that allow extending the MMT statement and module levels with new kinds of constructs. Then, sections and paragraphs (**P6a**), mathematical theories (**P6b**), structured declarations (**P5**) become concrete examples of such features. Moreover, recaps (**P8a**) and foundational ambiguity (**P8b**) can be represented more accurately.

Note that, with respect to the ideas from [Far08] our goal is to extend MMT's practical expressivity without extending its theoretical expressivity. This means preserving the central MMT properties and invariants while making it more suitable for building practical applications.

Remark 7. In the following we will use $\vec{*}$ to denote a sequence of elements $*_1, *_2, \ldots, *_k$.

5.1.1 Opaque Terms

We use opaque terms to represent mathematical expressions whose precise semantics is missing or incomplete. In practice this occurs in two common situations: 1. in a formal setting because

there are implicit parts left to be inferred by another process after parsing (e.g. type reconstruction), 2. in an informal setting because the primary representation is narrative and parts of the semantics cannot be inferred by systems such as natural language processors.

Conceptually, we represent them as terms where only some of their sub-terms (possibly none) have a known structure and semantics.

An **opaque term** $\bullet[\Gamma](\vec{E})$ is an IMMT object constructed from an IMMT context Γ and a sequence of IMMT terms $E_i \in \vec{E}$.

Intuitively, an opaque term $E_1 = \bullet[\Gamma](\vec{E})$ represents (any element from) the set of fully formal (i.e. with no opaque sub-terms) terms that bind the variable declarations from Γ and have each E_i as a sub-term – we call such a fully formal term *a formal grounding* of E_1. This idea originated in [Koh13].

Essentially, opaque terms are informal objects with zero or more formal sub-terms inside, under an optional context – since both the opaque or the partly non-opaque parts could bind variables. Therefore, opaque terms can be used to represent mathematical expressions whose semantics is unspecified or underspecified, as well as terms in an intermediate state of parsing or processing.

Before we dive into the inference system, we look at an example to solidify our intuitions.

Example 8. Consider the statement :

(5.1) *"It holds that $xy = md$ where m and d are the least common multiple and, respectively, greatest common divisor of x and y."*.

Assuming a trivial parser that does no significant natural language processing the resulting IMMT opaque term could be:

$$E_1 = \bullet[\cdot]((\mathit{equal}\ (\mathit{mul}\ x\ y)\ (\mathit{mul}\ m\ d)), m, d, x, y)$$

From the perspective of IMMT, E_1, represents any term that contains $xy = md$, m, d, x, and y as sub-terms. This, obviously, includes the intuitive meaning that mathematicians understand when reading that sentence. Therefore, the language allows for a potential, future processing step to refine the result to a concrete non-opaque term representing the intended meaning.

For instance, if a processing step can recognize the meaning of "*where*" as defining and binding m and, respectively d, the resulting term would be:

$$E_2 = \bullet[x, y, m = (\mathit{lcm}\ x\ y), d = (\mathit{gcd}\ x\ y)](\mathit{equal}\ (\mathit{mul}\ x\ y)\ (\mathit{mul}\ m\ d))$$

Inference System For defining the semantics of opaque objects, we add one new judgment to the ones used by MMT and presented in Section 4.2. The new judgment $\Theta \vdash^{\bullet}_{T} E$, shown in Figure 5.2, describes a term E that can be used as a sub-term to construct a well-formed opaque term.

Judgment	Intuition
$\Theta \vdash^{\bullet}_{T} E$	object E is a well-formed opaque sub-term in diagram Θ, over theory T

Figure 5.2: New Judgments for IMMT Objects

The key difference from regular terms is that the opaque parts are unknown to IMMT. They may contain binders that bind variables inside the sub-terms, possibly shadowing an outside context. Consider, for instance, the situation from Example 8 above where there are four free variables in the standalone opaque term. But x and y are meant to be bound by an external context, while m and d are defined and bound locally. This is unknown to IMMT without adequate math language parsing and processing. The rules from Figure 5.3 ensure that the resulting term is well-formed by effectively considering sub-terms of opaque terms as closed with the opaque parts providing the required bindings. This includes those that shadow existing bound variables.

$$\frac{\Theta, \mathit{free}(E) \vdash_T E}{\Theta \vdash^{\bullet}_{T} E}\ \bullet_{base}$$

$$\frac{\Theta, \cdot \vdash_T \Gamma \quad \Theta, \Gamma \vdash_T \Gamma_1 \quad \Theta \vdash^{\bullet}_{T} E_i \text{ for all } E_i \in \vec{E}}{\Theta, \Gamma \vdash_T \bullet[\Gamma_1](\vec{E})}\ \bullet_{inc}$$

Figure 5.3: Well formed IMMT Opaque Terms

The central object-level services that IMMT implements are (IMMT-level) equality and substitution. Foundations then implement typing and can refine equality by declaring (and implementing) their own equality relation. For opaque objects, IMMT-level *equality* is straightforward and defined recursively: $\bullet[\Gamma_1](\vec{E_1}) = \bullet[\Gamma_2](\vec{E_2})$ if and only if $\Gamma_1 = \Gamma_2$ and $\vec{E_1} = \vec{E_2}$.

The situation for *substitution* is different because of the potential implicit bindings that could occur in the opaque parts. Therefore, substitutions are not applied inside the sub-terms of the opaque objects. Instead, they are added as variable bindings to the context of the opaque term. Effectively, this has the consequence of delaying the substitution, which could be applied later as beta reduction if the opaque term is refined to a non-opaque one so that all internal binders become explicit. Concretely, substitution $[x/E_1]\bullet[\Gamma](\vec{E})$ of the term E_1 for the variable x in the opaque term $\bullet[\Gamma](\vec{E})$ is defined as $\bullet[x = E_1, \Gamma](\vec{E})$.

As expected, and already noticeable in the discussions above, opaqueness limits the potential for mechanization. In particular, formal processes such as type (or proof) checking or inference are generally impractical in the context of significant opaqueness. However, there are services for which even the partial information provided by the opaque terms is useful.

For instance, the sub-terms for opaque terms can be used for presentation, definition lookup (and expansion) or formula search. As is the theme throughout this thesis, the design of opaque terms

described above is informed and motivated by making such applications possible. We present the results from the perspective of the services and applications side in Section 8.2.2.

Note that there are other aspects that can be analyzed in the context of opaque terms. For instance, the *more formal than* relation $<:$ between two terms E_1 and E_2 could be formally defined. The intuition being that that $E_1 <: E_2$ if and only if any formal grounding of E_1 is also a formal grounding of E_2. Then operations on an opaque term such as adding a sub-term to the list or replacing a sub-term with another that contains it would produce a more formal term. However, we leave studying such relations for future work to be explored as applications appear that can make use of them.

5.1.2 Bipartite Terms

We use bipartite terms to represent structural dimorphism at the object level. Concretely, we add markup for annotating the content aspect of a mathematical object with its presentation. This is essentially a form of parallel markup inspired by MathML (see [ABC+10]) and we describe the details below.

For representing the content aspect we use the IMMT objects as described so far – which are basically MMT objects extended with opaque terms. However, for the presentational aspect we introduce new language primitives.

First, we define a dedicated construct for cross-referencing between the presentation and content aspects.

An object-level **realization specification** (or realization spec for short) is a tuple $\rho(p; pc)$ where p is the *position* of the referenced subterm and pc is the *presentation context* optionally providing language, format, and variant information for notation selection.

The position is a list of natural numbers viewing the term as a tree and representing the path from the root to the relevant node. Then, the empty list represents the entire term and each additional integer i takes the i-th child of the result. The presentation context is used by exporters to decide which notations will be prioritized to present the term. Then $\rho(p; pc)$ represents the presentation of the IMMT object referred to by p with presentation context pc. Therefore, the intuition is that a realization spec ρ specifies *how* IMMT exporters should realize the narration of a formal (sub)term.

We allow the presentation of objects to be any concrete format enhanced with realization specs. However, in practice the only such formats we encountered were XML (i.e. HTML or MATHML) and strings (e.g. LaTeX). Since XML subsumes strings, we will just use XML below and serialize realization specs into XML. For example, the listing below, shows a realization spec to the second child of the first child of the associated content object. It will resolve to the HTML rendering for the default verbalization in the German language.

```
1 <obj-real-spec position="1_2">
2   <pres-cont type="verbalization" language="german" format="HTML" variant="
      default"/>
```

```
3 </obj-real-spec>
```

The object **presentation** P^O for a XML-based presentation format f is an XML-language extending f with the iMMT primitives for realization specs (see listing above) under the namespace `http://kwarc.info/ns/iMMT`.

Therefore, presentation extends the native format f (e.g. HTML) with realiation specs. This implies that if resolving the references produces valid fragments of f, then the resulting document after resolving the references is a valid f-document. Note that for the common case of plain strings the extension becomes simply a sequence of string literals and realization specs $\rho(p; pc)$. In the following, we will use the string representation instead of the XML serialization where appropriate for brevity and simplicity.

Now we are ready to define bipartite terms.

A **bipartite term** $P^O \parallel E$ is an iMMT term representing a mathematical expression with both functional and narrative structure. Concretely, E represents its content (functional structure) and P^O its presentation.

One can see bipartite terms as regular MMT terms *annotated* with presentational information. Then the annotation can be either ignored or used by the relevant iMMT processes.

To fortify our intuitions about bipartite terms we revisit the statement from Example 8, this time by adding narrative information.

Example 9. For the statement:

> *"It holds that $xy = md$ where m and d are the least common multiple and, respectively, greatest common divisor of x and y."*.

Assuming the same content aspect as E_1 in Example 8, the resulting bipartite iMMT term could be:

$$P^O \parallel \bullet[\cdot]((equal\ (mul\ x\ y)\ (mul\ m\ d)), m, d, x, y)$$

where the presentation part P^O is:

> "*It holds that $\rho(1;\cdot)$ where $\rho(2;\cdot)$ and $\rho(3;\cdot)$ are the least common multiple and, respectively, greatest common divisor of $\rho(4;\cdot)$ and $\rho(5;\cdot)$.*"

Inference System For bipartite terms, we add one new judgement for well formed object-level realization specs. The new judgement $E \vdash^P \rho$, shown in Figure 5.4, describes a realization spec ρ that is well formed with respect to its associated content representation E. The corresponding rules for well formed realization specs are shown in Figure 5.5. While the rules look complex, the central idea is that a realization spec is well formed iff the position it points to exists in the content representation – i.e. in our case the term E. We also omit the straightforward rules for referencing inside the context of binders for simplicity. The intuition is that the position of a variable, type or definiens in the list of variable declarations forming the context is used to

Judgment	Intuition
$E \vdash^P \rho$	realization spec ρ is well formed with respect to corresponding content term E

Figure 5.4: Judgments for Bipartite Terms

resolve the realization specs. However, due to both the type and definiens being optional, the formal definitions quickly become verbose due to the number of cases.

$$\frac{}{E \vdash^P \rho(\cdot; pc)}\,\rho_{base} \qquad \frac{E_1 \vdash^P \rho(p; pc)}{E_1\ \Gamma.E_2 \vdash^P \rho(1, p; pc)}\,\rho_{\beta_{bind}}$$

$$\frac{\Gamma \vdash^P \rho(p; pc)}{E_1\ \Gamma.E_2 \vdash^P \rho(2, p; pc)}\,\rho_{\beta_{con}} \qquad \frac{E_2 \vdash^P \rho(p; pc)}{E_1\ \Gamma.E_2 \vdash^P \rho(3, p; pc)}\,\rho_{\beta_{scope}}$$

$$\frac{E_f \vdash^P \rho(p; pc)}{E_f\ \vec{E} \vdash^P \rho(1, p; pc)}\,\rho_{\mathcal{A}_{func}} \qquad \frac{E_i \vdash^P \rho(p; pc)}{E_f\ \vec{E} \vdash^P \rho(i, p; pc)}\,\rho_{\mathcal{A}_{args}}$$

$$\frac{E_i \vdash^P \rho(p; pc)}{\bullet[\Gamma](\vec{E}) \vdash^P \rho(i, p; pc)}\,\bullet_{args}$$

Figure 5.5: Well formed object-level Realization Specs

Based on the validity of realization specs we can now define well formed bipartite terms. The rules are shown in Figure 5.6 and ensure that the content aspect is well formed and that the realization specs are valid (i.e. that the referenced sub-terms exist). The intuition is therefore that an object presentation is well formed if all realization specs it contains are well formed with respect to its associated term.

$$\frac{\Theta, \Gamma \vdash_T E \quad E \vdash^P \rho_i \text{ for all } \rho_i \text{ in presentation } P_1^O}{\Theta, \Gamma \vdash_T P_1^O \parallel E}\,\parallel_{rule}$$

Figure 5.6: Well formed IMMT Bipartite Terms

IMMT level equality and substitution for bipartite terms are straightforward as they are defined based on the content aspect. Therefore, we define *equality* recursively as $P_1^O \parallel E_1 = P_2^O \parallel E_2$ if and only if $E_1 = E_2$. This is because IMMT equality evaluates the IMMT-level semantics of the object which are unaffected by the presentational annotations. As far as the semantic processes of IMMT are concerned, presentation is simply annotated meta-data.

Similarly, we define *substitution* $[x/E](P_1^O \parallel E_1)$ as $P_1^O \parallel ([x/E]E_1)$. Note that the rules for well formedness of realization specs are designed such that it is conserved by substitution.

Theorem 10. *If $E \vdash^P \rho$ (i.e. ρ is a valid realization spec w.r.t. term E) and γ is a valid substitution for E, then $[\gamma]E \vdash^P \rho$.*

The proof is elementary, given that every substitution replaces leaves in the term tree with subtrees, therefore *adding* new valid reference-able positions in the tree, but not removing any. The rules from Figure 5.5 do not permit referencing inside the context of opaque objects since that can be structurally changed by substitution (i.e. adding a new variable declaration) as defined above in Section 5.1.1.

5.1.3 Bipartite Declarations

To account for structural dimorphism at the declaration level we introduce a new structural element in IMMT which we call bipartite declaration. The intuition is that, like bipartite terms, bipartite declarations are constructed from two distinct parts, one to represent its narrative structure, and the other to represent its formal semantics.

In the following, we call the former the *pragmatic* declarations (or *pragmatics*) and the latter the *strict* declarations. The intuition is that the strict declarations correspond to the minimal language for representing the semantics that is easy to reason about (and that is largely backwards compatible with MMT). Meanwhile, pragmatics correspond to the rich, extensible layer of language constructs that human users (e.g. mathematicians) prefer.

These two aspects are typically conflated in mathematical representation languages, especially at the statement and module levels. Making them explicit in IMMT allows us to capture uniformly many commonly occurring phenomena from both formal languages and common mathematical language (CML).

We reuse pre-existing constructs from MMT wherever viable in order to achieve a minimal language that preserves as much as possible from MMT's properties and invariants. Concretely, we observe that both *documents* and modules are at their core named bodies (i.e. sets of statements). Similarly, in the presence of opaque and bipartite objects, narrative *paragraphs* as well as formal symbol declarations can both be represented using named IMMT terms (i.e. constants). Therefore, in IMMT we use constants to represent both narrative or content-side statements. Moreover, we use the notions of global name (URI) and body (set of statements) that are already used to form theories and views to also construct bipartite statements and modules. This is essential in achieving a minimal, scalable language that can be reasoned about and practically implemented. However, before we define bipartite declarations, we do need to introduce an additional primitive to IMMT.

An IMMT declaration-level **realization specification** (or realization spec for short) $l = \nu(g; nc)$ is a statement level construct with local name l referencing the declaration at global name g with narrative context nc.

Declaration-level realization specs are used for representing documents that refer to and build upon other documents (modules) and are effectively the narrative-side analog to imports. The narrative context describes how the narration should be realized by IMMT presenters. It contains information such as type (reference, inclusion, transclusion), sectioning level (paragraph, section, chapter, part, etc.), and title. The narrative context mirrors the presentation context of object-level realization specs but at the declaration level. It can be used by IMMT exporters (e.g. presenters) to produce browsable mathematical documents from the narrative parts of a module by resolving the references where needed.

Now, we are ready to define bipartite declarations. First, we differentiate between the statement and module level to preserve MMT's and OMDOC's three-level design. Therefore, to concretely realize bipartite declarations we introduce two new constructs into the IMMT language corresponding to the statement and, respectively, module level.

A **bipartite statement** B^S is an IMMT statement $l = [\![\vec{S_1} \parallel \vec{S_2}]\!]$, where l is its *name* and $\vec{S_1}$ are its *pragmatic statements* represented as a sequence of either realization specs, constants, or other bipartite statements. Similarly, $\vec{S_2}$ are its *strict statements* represented as a sequence of either constants or includes.

A **bipartite module** B^M is an IMMT module $l = [\![\vec{D_1} \parallel \vec{D_2}]\!]$, where l is its *name* and $\vec{D_1}$ are its *pragmatic modules* represented as a sequence of either pragmatic statements, theories, views or other bipartite modules. Similarly, $\vec{D_2}$ are its *strict declarations* represented as a sequence of theories or views.

The XML serialization for the bipartite statements is shown in the listing below where each of the "..." represents the (recursive) XML serialization of the respective IMMT elements. The XML for bipartite modules is analogous except the root tag is `bipartite-mod`.

```
<bipartite-stm name="...">
  <pragmatics>
    ...
  </pragmatics>
  <stricts>
    ...
  </stricts>
</bipartite-stm>
```

Bipartite statements and modules scale the bipartite design, used for bipartite objects and inspired from parallel-markup, to all three levels of IMMT. A key difference between bipartite statements and modules are the kinds of declarations that they can be composed from. Bipartite modules can additionally contain modules in their strict declarations as well as other, nested, bipartite modules in their pragmatics. With respect to the pre-existing MMT constructs, bipartite statements can only occur inside theories (since they are statement-level elements) while bipartite modules can only occur inside a diagram.

Note that, in this thesis, we do not cover the case of bipartite declarations in views. They are redundant because views represent a relation between the semantics (i.e. strict declarations) of theories so they can ignore the narrative structure encoded by bipartite declarations. Instead,

views provide a constant declaration for any undefined constant in the strict declarations of a bipartite statement from its domain. We clarify this below in Section 5.1.3 where we discuss the inference system. However, note that it is possible (and often convenient) for views to contain such structured declarations so that bipartite statements in the source theory can be mapped more naturally to bipartite statements in the target theory. We discuss that as part of future work in Chapter 9.

For bipartite declarations, we require names so that they can be uniquely referenced later. We assume that if no name is present in the source document, names will be automatically generated when importing content into iMMT. Moreover, in the following, if a name is unreferenced or otherwise irrelevant we will use the symbol "_" to denote it. We leave the rest of the discussion on iMMT URIs and referencing declarations to Section 5.2.2 below.

Bipartite declarations are useful because often there is not a one-to-one correspondence between narrative text and content expressions which can be represented via bipartite terms. Instead, one piece of narrative text can correspond to one or more declarations in the content representation. Conversely, one declaration in the content representation can correspond to one or more phrases or paragraphs such as large structured definitions or long proofs.

Example 11. Consider, for instance Example 2.23 mentioned above which we repeat below :

"***Theorem 7.1**: There is exactly one solution (called the **exponential function** $f(x) = e^x$) to the equation $f' = f$ with $f(0) = 1$.*"

A corresponding bipartite declaration is $Theorem7.1 = [\![\_ = E \parallel C_{thy}, C_{exp}]\!]$ where E is $\|P^O\cdot$ and encodes the presentation for the theorem as shown above. Moreover, C_{thy} and C_{exp} are:

$$thy : \Vdash \exists^! f : \mathbb{R} \to \mathbb{R}.f\ 0 \doteq 1 \wedge \forall x : \mathbb{R}.f'\ x \doteq f\ x$$
$$exp : \mathbb{R} \to \mathbb{R} = \iota\ (\lambda\ f : \mathbb{R} \to \mathbb{R}.f\ 0 \doteq 1 \wedge \forall x : \mathbb{R}.f'\ x \doteq f\ x)\ thy$$

where we assume the required concepts are already defined in an included theory. Concretely, $\Vdash$ is the operator for the type of proofs, $\exists^!$ is unique existence, $\mathbb{R}$ is the type of real numbers, $\doteq$ is equality between reals, ι is the definite description operator, and $\forall$ is the universal quantifier.

For bipartite modules, a common example is an isomorphism between two (or more) theories S and T which would correspond to a pair of views $v_1 : S \to T$ and $v_2 : T \to S$. A related, concrete example is shown in Example 2.40 where, for instance, the following statement occurs in the original text (from [Bar15]) "*There are several equivalent ways to define multinets. Here we present them using pencils of plane curves.*". We discuss this in more details in Section 6.1.4 below.

Inference System We do not need additional judgments for bipartite declarations as we can reuse the judgments for valid statements and, respectively, modules and just add the appropriate rules.

In order to reference declarations inside the new constructs (i.e. bipartite statements and modules) we need to extend the URIs for iMMT. We discuss this in more detail in Section 5.2.2 below, but the intuition is that we use two separators to look inside bipartite declarations: "!" for

the pragmatics and "?" for the strict declarations.

We give the rules for bipartite declarations in Figures 5.7 and 5.8. The former shows the additional rules establishing when adding a declaration to a (theory in a) diagram preserves well-formedness. The latter, formalizes the visibility (access) rules for bipartite declarations. We discuss each individually below.

First, the rule $B^S_{base\_add}$ states that adding a bipartite statement with no pragmatics (to a theory in a diagram) preserves well-formedness if adding each of its strict statements sequentially (to that theory) preserves well-formedness. This follows the intuitions discussed above, that the strict statements represent its semantics as seen from the containing theory. As expected, the rule additionally requires that the referenced theory exists and that global name introduced by the bipartite statement is fresh.

Then, the rule $B^M_{base\_add}$ is the direct, module-level analogue, stating that adding a bipartite module with no pragmatics (to a diagram) preserves well-formedness if adding each of its strict modules sequentially (to that diagram) preserves well-formedness. Similarly, it requires that the newly introduced global names are fresh.

The last two rules formalize the conditions for adding pragmatics to existing bipartite declarations. $S^N_{prag\_add}$ states that adding a realization spec to the pragmatics of a bipartite declaration preserves well-formedness if the bipartite declaration exists, the referenced (module) global name exists and the local name of the realization spec is fresh. Similarly, $S^C_{prag\_add}$ states that adding a constant (to the pragmatics of a bipartite declaration) preserves well-formedness, if the bipartite declaration exists, the local name of the constant is fresh, and that the optional type and definition of the constant are well-formed with respect to the bipartite declaration.

The last conditions for $S^C_{prag\_add}$ involve well-formedness of objects which in turn involve the visibility conditions between global names (see Section 4.2). Concretely, for an object to be well-formed with respect to a global name g it is required that every global name g_i it references is visible from g. This is formalized by the visibility judgment $\Theta \vdash^v_g g_i$. The visibility rules for bipartite declarations are shown in Figure 5.8. First, B_{vis_strict} states the pragmatics of a bipartite declaration can access its strict declarations. Concretely, any global name visible for the strict declarations is also visible for the pragmatics. Then, B_{vis_prag} states that realization specs make visible locally all declarations that are visible in their target.

Corollary 1. *By removing the narrative information from a well-formed* iMMT *diagram* Θ *(i.e. replacing every bipartite declaration with its strict declarations) we obtain a well-formed* MMT *diagram.*

The proof is straightforward following the rules from Figure 5.7. But, the statement above is essential because it means that the central MMT algorithms that ignore narration can still work for iMMT. This is consequential for both the theoretical aspect (reasoning about the language) and the implementation.

$$\frac{\vdash \Theta(g) \Rightarrow T \quad \vdash \Theta(g!l) \Rightarrow \lightning \quad \Theta \vdash^+_g \vec{S}}{\Theta \vdash^+_g l = [\![\cdot \parallel \vec{S}]\!]} B^S_{base\_add}$$

$$\frac{\vdash \Theta(g) \Rightarrow \lightning \quad \Theta \vdash^+ \vec{M}}{\Theta \vdash^+ g = [\![\cdot \parallel \vec{M}]\!]} B^M_{base\_add}$$

$$\frac{\vdash \Theta(g!l) \Rightarrow l = [\![\Sigma_1 \parallel \Sigma_2]\!] \quad \vdash \Theta(g^M) \Rightarrow M \quad \vdash \Theta(g!l!l_1) \Rightarrow \lightning}{\Theta \vdash^+_{g!l} l_1 = \nu(g^M; nc)} S^N_{prag\_add}$$

$$\frac{\vdash \Theta(g!l) \Rightarrow l = [\![\Sigma_1 \parallel \Sigma_2]\!] \quad \vdash \Theta(g!l!c) \Rightarrow \lightning \quad [\Theta, \cdot \vdash_{g!l} E_1] \quad [\Theta, \cdot \vdash_{g!l} E_2]}{\Theta \vdash^+_{g!l} c[:E_1][=E_2]} S^C_{prag\_add}$$

Figure 5.7: Well Formed IMMT Bipartite Declarations

$$\frac{\vdash \Theta(g!l) \Rightarrow l = [\![\Sigma_1 \parallel \Sigma_2]\!] \quad \Theta \vdash^v_g g^s}{\Theta \vdash^v_{g!l} g^s} B_{vis\_strict}$$

$$\frac{\vdash \Theta(g!l) \Rightarrow l = [\![\Sigma_1 \parallel \Sigma_2]\!] \quad \vdash \Sigma_1(\_) \Rightarrow l_1 = \nu(g^M; nc) \quad \Theta \vdash^v_{g^M} g^S}{\Theta \vdash^v_{g!l} g^S} B_{vis\_prag}$$

Figure 5.8: Visibility Rules for Bipartite Declarations

5.1.4 Derived Declarations

The binary design of bipartite declarations described above can do more. It allows uniformly modeling and adequately representing the pragmatics of structurally complex declaration containers such as records or structured proofs (with inline lemmas, sub-proofs, etc.). We model this as the case where the strict declarations of a bipartite declaration are practically derivable (or computable) from its pragmatics in a process we call *elaboration*. Note that the practicality of elaboration is not a fixed point but is relative to the efficiency of algorithms and, in the informal case, the capability of natural language processing and understanding.

To represent such declarations we add two new constructs to IMMT: structural features and derived declarations. Note that the idea of pragmatic extensions for the MMT language is not new as [HKR12] introduced an extension language for the statement level. It provided syntactic means for defining pragmatic language features and their semantics in terms of strict MMT. However,

here, we give a much more expressive extension mechanism which 1. is not syntactically constrained and instead lets language designers provide arbitrary elaboration functions (see below), 2. allows extending *both* the statement and module levels, 3. is built on top of the dual-container design of bipartite declarations.

A **structural feature** F of a theory $g = \Sigma$ is a tuple $\langle \mathfrak{f}, g, n, \mathcal{V}, \mathcal{E} \rangle$ where $\mathfrak{f}$ is its *name* and $n \in \mathbb{N}$ is its *arity*. Then $\mathcal{V}$ is the *validity predicate* which evaluates tuples of the form $\langle l, \vec{E}, \vec{D} \rangle$. Finally, $\mathcal{E}$ is the *elaboration function* which maps a tuple $\langle l, \vec{E}, \vec{D} \rangle$ to a sequence of declarations $\vec{D}$.

The intuition behind structural features is to allow IMMT theories to declare their own language features. This is often useful because IMMT theories can be used to define other languages such as logics or logical frameworks. Such languages may have their own declaration-level constructs (e.g. records, sections, structured proofs, etc.) which can now be explicitly defined in IMMT as structural features of their respective theory. The features of a theory T can then be used to instantiate concrete IMMT declarations in modules that extend T.

We introduce a new construct to represent such instances which we call derived declarations. As for bipartite declarations, we distinguish between the statement and module levels.

A **derived statement** D^S is a 4-tuple $\langle l, \mathfrak{f}, \vec{E}, \vec{S} \rangle$ (written $l = \mathfrak{f}\ \vec{E}\ \{\vec{S}\}$), we call l its *name*, $\mathfrak{f}$ its *feature*, the E_i its *arguments*, and the S_i its *pragmatic statements*.

A **derived module** D^M is a 5-tuple $\langle g, g^T, \mathfrak{f}, \vec{E}, \vec{D} \rangle$ (written $g : g^T = \mathfrak{f}\ \vec{E}\ \{\vec{D}\}$), we call g its *global name*, g^T is (the global name of) its *meta theory*, $\mathfrak{f}$ its *feature*, the E_i its *arguments*, and the D_i its *pragmatic declarations*.

Derived declarations are part of the IMMT grammar, and the same general syntax is used for them regardless of their feature. Thus, we can introduce almost arbitrary new features and still have algorithms that treat them uniformly. Concretely, the XML serialization for derived statements is shown in the listing below where each of the "..." represents the (recursive) XML serialization of the respective IMMT elements. The XML for derived modules is analogous except the root tag is `derived-mod` and it has an additional `meta-theory` attribute.

```
1 <derived-stm name="..." feature="...">
2   <arguments>
3     ...
4   </arguments>
5   <pragmatics>
6     ...
7   </pragmatics>
8 </derived-stm>
```

The intuition of derived declarations is that their feature expands derived declarations to bipartite ones by using the elaboration function to produce the strict declarations (by applying it to the tuple $\langle l, \vec{E}, \vec{D} \rangle$ formed from the name, arguments and pragmatics of the derived declaration). Moreover, the validity predicate is analogously used to define the well-formedness of derived declaration. Note that the dual case when the pragmatics can be inferred from the strict declaration is possible but outside the scope of this thesis.

Before we give the judgments and inference system below we give a few examples to solidify our intuitions. Structural features are useful to define a wide array of new kinds of declaration-level constructs for representing knowledge. These include narrative constructs such as documents, sections or comments, content ones such as inductive datatypes or MMT structures, and mixed ones such as definitions, proofs and theorems.

Example 12. Consider Example 11, where a single statement is both a theorem (of unique existence) and a definition of that concept. This can be generically defined as a structural feature $\langle \mathfrak{DefTheorem}, g, 2, \mathcal{V}, \mathcal{E} \rangle$. Then, theories importing g can use the feature and declare derived statements such as $l = \mathfrak{DefTheorem}\ E_1, E_2\ \{\vec{S}\}$. The validity predicate $\mathcal{V}$ checks the following:

1. there are indeed 2 arguments
2. the first argument E_1 is a proposition stating a unique existential
3. the second argument E_2 is a proof of E_1 (i.e. E_2 has type $\Vdash E_1$)
4. the statement list $\vec{S}$ is empty.

The elaboration function $\mathcal{E}$ produces two statements: $l/thy : \Vdash E_1 = E_2$ and $l/def = \iota\, E_2$ where ι is the definite description operator. Therefore, we can systematically capture in IMMT both the intended semantics (the two statements in the elaboration) as well as the narrative structure. In practice, language designers would identify the structural features of their domain of application and implement them in IMMT. Then, the intended knowledge can be represented in IMMT in a structure-preserving way.

We present some example features that recover and extend the original expressivity of MMT within IMMT below in Chapter 6. Moreover, in Chapter 7 we discuss library imports and define a series of features for adequately representing the primitive constructs for each base language of the imported libraries.

Inference System The additional judgments that we need to define the semantics of derived declarations are given in Figure 5.9.

The key novelty are the judgments for elaboration. $\Theta \vdash_g D^S \rightsquigarrow \vec{S}$ defines the result of elaborating a derived statement D^S using the elaboration function implemented by its feature. Similarly, $\Theta \vdash D^M \rightsquigarrow \vec{M}$ defines the result of elaborating a derived module D^M using the elaboration function implemented by its feature. Additionally, the judgment $\vdash^{\mathfrak{f}} D$ states that the declaration D satisfies validity constraints imposed by the feature $\mathfrak{f}$. Concretely, assume $\mathcal{V}$ is the validity predicate of $\mathfrak{f}$ and D has local name l, arguments $\vec{E}$ and pragmatics $\vec{S}$. Then $\mathcal{V}$ holds for $\langle l, \vec{E}, \vec{S} \rangle$.

Now we can give the inference rules which are shown in Figure 5.10. The intuition is that a derived declaration is well formed if the corresponding bipartite declaration constructed from its pragmatics and its elaboration is well formed. Additionally, we require that the validity function as implemented by the feature holds. Therefore, the rules follow the intuition that derived declarations plus structural features are essentially syntactic sugar and macros built on top of bipartite declarations.

Judgment	Intuition
$\Theta \vdash_g D^S \rightsquigarrow \vec{S}$	derived statement D^S elaborates to $\vec{S}$ in the module at g
$\Theta \vdash D^M \rightsquigarrow \vec{M}$	derived module D^M elaborates to $\vec{M}$ in diagram Θ
$\vdash^{\mathfrak{f}} D$	the validity predicate of the feature $\mathfrak{f}$ holds for the derived declaration D

Figure 5.9: Judgments for IMMT Derived Declarations

$$\frac{\vdash^{\mathfrak{f}} l = \mathfrak{f}\ \vec{E}\ \{\vec{S_1}\} \quad \Theta \vdash_g l = \mathfrak{f}\ \vec{E}\ \{\vec{S_1}\} \rightsquigarrow \vec{S_2} \quad \Theta \vdash^+_g l = [\![\vec{S_1} \parallel \vec{S_2}]\!]}{\Theta \vdash^+_g l = \mathfrak{f}\ \vec{E}\ \{\vec{S_1}\}}\ D^S_{add}$$

$$\frac{\vdash^{\mathfrak{f}} g : g^T = \mathfrak{f}\ \vec{E}\ \{\vec{D}\} \quad \Theta \vdash g : g^T = \mathfrak{f}\ \vec{E}\ \{\vec{D}\} \rightsquigarrow \vec{M} \quad \Theta \vdash^+ g = [\![\vec{D} \parallel \vec{M}]\!]}{\Theta \vdash^+ g : g^T = \mathfrak{f}\ \vec{E}\ \{\vec{D}\}}\ D^M_{add}$$

Figure 5.10: Well Formed IMMT Derived Declarations

Theorem 13. *By replacing every derived declaration from a well-formed* IMMT *diagram* Θ *with its corresponding bipartite declaration after elaboration we obtain a well-formed* IMMT *diagram.*

The proof is trivial since this is exactly how the well-formedness of derived declarations is defined in Figure 5.10.

Together with Theorem 1 this means that a well-formed IMMT diagram with derived declarations can also be reduced to a well-formed MMT one by ignoring narration. Again, this is essential for reasoning about the language as well as for its implementation. Crucially, it means that we can safely extend the MMT system to produce an implementation for IMMT.

5.2 The IMMT Language and System

In this section we present the IMMT language by aggregating the extensions described above together into one language. We present the grammar below in Section 5.2.1, then discuss IMMT URIs and referencing declarations in Section 5.2.2. Finally, we present the implementation of the language and the resulting IMMT system in Section 5.2.3.

5.2.1 Grammar

The IMMT grammar is shown in Figure 5.11 with the original MMT primitives grayed out. Each of the individual extensions are discussed above in Section 5.1. The central idea is that we

preserve MMT's three-level design by having explicit, object, statement and module levels but extend expressivity at each level.

At the object level we extend the term language by adding *opaque terms* (see 5.1.1) and *bipartite terms* (see 5.1.2). The former allows representing in IMMT fully or partially opaque knowledge. The latter allows capturing structural dimorphism at the object level. Since structural dimorphism occurs at all levels, we add corresponding constructs to the statement and module level.

Concretely, at the statement level we introduce *bipartite statements* (see 5.1.3) to represent both the internal structure and the semantics of complex statements. The structure of a bipartite statement, which we call its pragmatics, is represented as a sequence of *pragmatic statements*, and its semantics is represented as a sequence of *strict statements*. Moreover, in order to realize extensibility at the statement level in IMMT we add *structural features* (see 5.1.4) to the IMMT statement level. They allow language designers to declare their own set of declaration level constructs. At the statement level these constructs can be instantiated as *derived statements*. They capture the structure of the instances as pragmatic statements (same as bipartite declarations). However their semantics (represented as strict statements) is derived by their structural feature in a process we call *elaboration* (see 5.1.4 for details).

At the module level, the IMMT extensions mirror the ones at the statement level. First we add *bipartite modules* for structural dimorphism (see 5.1.3). The central difference is that their semantics is represented as *strict modules* and their pragmatics can be either pragmatic statements or modules (theories, views or bipartite modules). We also add *derived modules* (see 5.1.4) to realize feature-based extensibility at the module level. They behave analogously to derived statements with respect to elaboration, but being at the module level, their structure and semantics are represented as pragmatic and, respectively, strict modules.

5.2.2 Referencing Declarations

In addition to the object statement and module levels we also extend MMT URIs. We set as a requirement that every IMMT declaration has its own global name (URIs). This is essential to allow declarations from an IMMT diagram to be used (referenced) by new content as well as by services and applications. In fact this is a central motivation for the declaration-level extensions that we presented above. Human users (e.g. mathematicians) use narrative constructs as part of doing, writing, or talking about mathematics. So, having these narrative structures represented and reference-able is a prerequisite for being able to build a wider range of semantic, user-facing, practical applications.

Compared to MMT, the problem is that for the new bipartite declarations there are two containers in parallel, namely for the pragmatic and strict declarations. Therefore, MMT URIs which use the "?" separator for *looking into* containers such as theories or views do not work directly. In IMMT we need more than one separator since we can have two containers that need to be accessed for one declaration. This is because for bipartite declarations *looking into* requires a choice, namely which one of the two containers to access. Therefore, we use two separators for global names

Declaration	D	::=	$M \mid S$
Diagram	Θ	::=	M^*
Body	Σ	::=	S^*
Module Level			
Module	M	::=	$T \mid V \mid B^M \mid D^M$
Strict Module	R^M	::=	$T \mid V$
Pragmatic Mod.	P^M	::=	$P^S \mid M$
Theory	T	::=	$g = \{\Sigma\}$
View	V	::=	$g : g \to g = \{\Sigma\}$
Bipartite Mod.	B^M	::=	$g = [\![P^{M*} \parallel R^{M*}]\!]$
Derived Mod.	D^M	::=	$g : g = \mathfrak{f}\ E^*\ \{P^{M*}\}$
Statement Level			
Statement	S	::=	$C \mid I \mid F \mid B^S \mid \nu$
Strict Stm.	R^S	::=	$C \mid I$
Pragmatic Stm.	P^S	::=	$C \mid F \mid B^S \mid \nu$
Constant	C	::=	$l[:E][=E]$
Include	I	::=	`include` g
Bipartite Stm.	B^S	::=	$l = [\![\vec{P^S} \parallel \vec{R^S}]\!]$
Decl. Realization Spec.	ν	::=	`see Section 5.1.3`
Struct. Feature	F	::=	$\mathfrak{f}\ cp$
Derived Stm.	D^S	::=	$l = \mathfrak{f}\ \vec{E}\ \{\vec{P^S}\}$
Object Level			
Object	O	::=	$E \mid \Gamma$
Expression	E	::=	$l \mid g \mid E\ E^* \mid E\ \Gamma.E \mid \bullet[\Gamma](E^*) \mid P^O \parallel E$
Context	Γ	::=	$[(l[:E][=E])^*]$
Obj. Realization Spec.	P^O	::=	`see Section 5.1.2`
Identifiers			
Global name	g	::=	$URI \mid g?l \mid g!l$
Local name	l	::=	`String`
Feature name	$\mathfrak{f}$	::=	`String`
Class Path	cp	::=	`Java ClassPath`

Figure 5.11: The iMMT Grammar

instead: "!" and "?" as shown in the grammar from Figure 5.11. The intuition is that "!" is used to access narrative structures (i.e. pragmatics) and "?" is used to access declarations that represent semantics (e.g. theories, views, constant or strict declarations). Therefore, the URIs for pre-existing MMT primitives remain unchanged and use the "?" separator as before. This means that in situations (e.g. type-checking, proof-inference) where the narration is irrelevant we preserve the central properties and invariants of MMT. This is practically useful for preserving the correctness of such algorithms and also for backwards compatibility of the systems and libraries.

The new rules are for the URIs of bipartite declarations are shown in Figure 5.12. They formalize URI-based lookup for bipartite declarations in diagrams and follow the intuitions given above.

The first rule, B_{uri_strict} covers the case for referencing strict declarations and states that they can be accessed by using the URI of their container and their local name, separated by "?". Concretely, if the container at URI g includes a bipartite declaration which in turn contains a strict declaration D with name l, then D can be referenced with URI $g?l$.

The last two rules cover the case for accessing pragmatic declarations and state that pragmatics can be referenced using the URI of their container together with their local name, separated by "?". Since bipartite declarations can themselves be pragmatics (and therefore nest) we formalize this using two rules: $B_{uri\_prag\_base}$ for pragmatics directly inside theories, and $B_{uri\_prag\_rec}$ for pragmatics within other bipartite declarations.

$$\frac{\vdash \Theta(g!l) \Rightarrow l = [\![\Sigma_1 \parallel \Sigma_2]\!] \quad \vdash \Sigma_2(l_2) \Rightarrow D}{\vdash \Theta(g?l_2) \Rightarrow D} B_{uri\_strict}$$

$$\frac{\vdash \Theta(g) \Rightarrow g = \Sigma \quad \vdash \Sigma(l) \Rightarrow l = [\![\Sigma_1 \parallel \Sigma_2]\!]}{\vdash \Theta(g!l) \Rightarrow l = [\![\Sigma_1 \parallel \Sigma_2]\!]} B_{uri\_prag\_base}$$

$$\frac{\vdash \Theta(g!l) \Rightarrow l = [\![\Sigma_1 \parallel \Sigma_2]\!] \quad \vdash \Sigma_1(l_1) \Rightarrow D}{\vdash \Theta(g!l!l_1) \Rightarrow D} B_{uri\_prag\_rec}$$

Figure 5.12: URIs for IMMT Bipartite Declarations

5.2.3 Implementation

We base our implementation on the MMT system described in Section 4.2. We extend the implementation to add the constructs described above and integrate it into the existing implementation. The code is available at `https://github.com/KWARC/MMT` in the `src/mmt-api/trunk` sub-project. Bipartite and derived declarations are added as new statement-level classes and are implemented as described in Section 5.1.3 and 5.1.4.

Structural features are implemented as a new class of IMMT plugins which must implement the validity predicate and elaboration function as specified above. However, for adding them to a theory, we do not add a new primitive to the code-base, but instead reuse the infrastructure for MMT *rules* described in [Rab15a]. Rules are new primitive statements which takes as an argument a string literal pointing to a Scala class implementing an IMMT/MMT plugin. Therefore, they allow enabling such plugins as part of declaring theories. It is mainly used for defining type-checking and well-formedness rules for constants declared in foundations (such as λ or Π for the LF logical framework [Pfe91]). But it applies directly for enabling structural features in a theory by adding a rule referencing the Scala class that implements the feature. By being user-definable and integrated in the module system, structural features represent the key construct that allows modular, declaration-level extensibility in both the IMMT language and system.

On the other hand, bipartite and opaque objects are not added as primitives and instead encoded using existing MMT object constructors in the implementation. Concretely, we can add a dedicated foundation that declares symbols corresponding to the primitives introduced above and implement complementary IMMT plugins that implements the well-formedness rules as IMMT/MMT rules. The resulting theory is shown in Figure 5.13.

From that perspective, the grammar introduced in Section 5.2.1 above can be considered just syntactic sugar that gets expanded to regular MMT objects as long as the theory `iobjects` is in scope (i.e. transitively included). Note that, in MMT, the flexiformal foundation includes both the flexiformal objects foundation `iobjects` and the collection of necessary structural features (discussed below in Chapter 6).

To cover the fact that an opaque term $\bullet[\Gamma](\vec{E})$ works as both a binder and an application from the perspective of MMT/OPENMATH primitives we add two constants for opaque terms instead of one. They correspond to the application and, respectively, a binding operation. Therefore, we expand $\bullet[\Gamma](\vec{E})$ to $(\bullet_\beta\ \Gamma.(\bullet_{\mathcal{A}}\ \vec{E}))$.

theory `iobjects` $= \{$
  constant $\bullet_\beta$
  constant $\bullet_{\mathcal{A}}$
  rules $\bullet_{base}, \bullet_{inc}$
  constant $\|$
  rules $\|_{rule}, \|_\rho$
$\}$

Figure 5.13: Foundation for Flexiformal Objects in MMT

For parallel markup, we use MMT/OPENMATH attributions [BCC+04, Rab14b] which we omitted here for simplicity but are implemented in the MMT system. Attributions $@(E, g, v)$ are effectively a new primitive constructor for terms that allows annotating a term E using a key g with arbitrary data v. We use attributions to annotate IMMT terms with presentational information to represent natively bipartite terms. The constant $\|$ from Figure 5.13 is the attribute key, the n arration is the value and the content aspect is the term being annotated. Therefore, an IMMT term $P^O \parallel E$ is represented as $@(E, \|, P^O)$.

Chapter 6

Representing Flexiformal Phenomena in IMMT

In this Chapter we evaluate the IMMT language by applying it to represent common constructs from both formal and informal mathematics. Effectively, we take the language designer perspective and try to design IMMT foundations for narrative mathematics by defining structural features capable of representing the phenomena observed in Chapter 2. First, in Section 6.1, we look at the informal phenomena, specifically those that the MMT language cannot represent adequately. Then, in Section 6.2, we select and analyze some common constructs used in formal systems. In both situations, we discuss each construct individually and examine how to represent it in IMMT.

Before describing individual features, we introduce some auxiliary definitions. In the IMMT implementation, these are provided as utility functions.

Definition 14 (Renaming)**.** For a partial function r from names to names and a body Σ, we write $[r]\Sigma$ for the result of replacing in Σ every occurrence of c with $r(c)$ as follows:

- every declaration named c with the corresponding declaration named $r(c)$,
- every expression E with the expression obtained by replacing every occurrence of a name c or c/l in E with $r(c)$ or $r(c)/l$, respectively.

We write $[c/\_]\Sigma$ for $[\ldots, n \mapsto c/n, \ldots]\Sigma$ where the ellipsis runs over the names of all declarations in Σ.

Renaming does exactly what the name suggests, it updates the names of the declarations in Σ following the renaming function r. Additionally, it looks into expressions and updates any references to the renamed declarations to the new name to make sure well-formedness is preserved.

Definition 15 (Translation)**.** For a partial function t from names to expressions, we write $\bar{t}(E)$ for the result of replacing in E every occurrence of a name c for which $t(c) = E'$ with E'.

Moreover, we use the term **subfeature** for a structural feature $\mathfrak{sf}$ that is declared with the explicit purpose of appearing as an pragmatic declaration in a *parent* elaborating feature $\mathfrak{f}$. The elaboration of instances of $\mathfrak{sf}$ is then defined as part of the elaboration function of $\mathfrak{f}$.

As we will see below, defining a structural feature sometimes involves defining other, auxiliary features that are only needed to occur as pragmatics within instances of their parent feature. Examples include reasoning steps in proofs, operations in mathematical structures, or isomorphisms in establishing common ground (as seen in 2.1). Since instances of the subfeatures always occur inside their parent feature, it is often more convenient to describe (and implement) their elaboration together. Therefore, the elaboration of subfeatures is usually defined using extra cases within the elaboration function of their parent.

6.1 Informal Mathematics

In this Section we discuss representing informal mathematical libraries in IMMT. Since there is no automatic tool for semantization of informal mathematics, we cannot do a meaningful import of a large informal corpus (i.e. one that actually enables useful practical applications). Indeed, we see our work on IMMT as the basis on which such an import could be developed. Instead, we systematically address the phenomena from Section 2 and present how they can be represented in IMMT. In essence, we are building a target format in IMMT for representing informal mathematics. Specifically, we define structural features for each construct to test the expressivity of IMMT with respect to informal libraries in common mathematical language.

Naturally, we focus only on those phenomena that MMT does not natively support . As discussed in Section 3, **P1** Formulae as Primitive Objects, **P1b** Naming Objects Locally, **P1c** Referential Meaning and **P7** Framing are already covered by plain MMT. Moreover, **P9** Structural Dimorphism, **R1** Opaqueness were discussed in Section 5. Therefore, here, we discuss representing **P3** Notations and Verbalizations, **P5** Structured Declarations, **P6b** Dynamic Theories, **P8a** Recaps and **P8b** Foundational Ambiguity.

Before we demonstrate how to represent each of these phenomena below, we develop an *informal foundation* IF in IMMT based on the logical framework LF (which provides the primitives for `type` and for the binders λ and Π). IF will serve as the logical foundation for the examples in the rest of this chapter. It is inspired by the Mizar logic [Wie07a, Ban03] and Weak Type Theory (WTT) [KN04]. Concretely, it is similar to the LF formalization of the Mizar base logic described in [IR11]. The key idea of IF is a foundation based on first-order logic and ZFC together with a notion of weak types that can be constructed via predicates. We use the usual notations in declaring the types and terms for convenience and readability, but only discuss how to represent these notations in IMMT below in Section 6.1.1.

Note that we make no particular claims about the expressivity of IF so far and just use it as an example. Instead we rely on the foundation independence of MMT (and therefore IMMT) to allow mathematicians and library developers to use (or design) an appropriate foundation for each individual development or library. However, in the following, we extend IF with appropriate

theory $IF : LF = \{$

$prop$: `type`
set : `type`
wtp : `type`
$is : set \to wtp \to prop$
$\Vdash : prop \to$ `type`

$\wedge : prop \to prop \to prop$
$\neg : prop \to prop$
$\vee : prop \to prop \to prop = \lambda\, a : prop.\lambda\, b : prop.\neg(\neg a \wedge \neg b)$
$\Rightarrow : prop \to prop \to prop = \lambda\, a : prop.\lambda\, b : prop.\neg a \vee b$
$\Leftrightarrow : prop \to prop \to prop = \lambda\, a : prop.\lambda\, b : prop.a \Rightarrow b \wedge b \Rightarrow a$
$\forall : (set \to prop) \to prop$
$\exists : (set \to prop) \to prop = \lambda\, P : set \to prop.\neg\forall x.\neg P\ x$
$\wedge_I : \Pi\, A : prop.\Pi\, B : prop.\ \Vdash A \to \Vdash B \to \Vdash A \wedge B$
$\wedge_{El} : \Pi\, A : prop.\Pi\, B : prop.\ \Vdash A \wedge B \to \Vdash A$
$\vdots$
$wtp_I : \Pi\, T : wtp.\Pi\, P : set \to prop.wtp$
$wtp_E : \Pi\, T : wtp.\Pi\, P : set \to prop.\ \Vdash \forall_{x:set} x\ is\ T \wedge Px \Leftrightarrow x\ is\ wtp_I\ T\ P$
$\vdots$
$extensionality : \forall\, x : set.\forall\, y : set.(\forall\, z : set.z \in x \Leftrightarrow z \in y) \Rightarrow x = y$
$\emptyset : set$
$\vdots$

$\}$

Figure 6.1: Informal Foundation for IMMT

structural features to capture the outstanding phenomena discussed above. Therefore, we produce the blueprint for an informal foundation that could be used to represent both the narrative structures and semantics of informal mathematics. Again, while acknowledging the limitations of IF as presented here, we see it as a basis for practically realizing a meaningful import of an informal corpus in the future.

6.1.1 Notations and Verbalizations

Notations are an important part of both formal and informal mathematics. They act as rules for transforming between the content and presentation forms and can, therefore, be used for both parsing and presenting. Unlike MMT, in IMMT we do not add explicit productions for notations. Instead, we represent notations as a structural feature where instances give the content-presentation relation as a bipartite term.

This has two advantages in addition to keeping the core language simple. First, by treating notations as standalone declarations we leverage the modularity of IMMT to manage scoping and re-usability of notations. Second, by using bipartite terms to represent notations, we leverage the genericity and expressivity of the IMMT object level to obtain a powerful and extensible notation language.

We define a special nullary structural feature $\mathfrak{Notation}$ whose elaboration is the constant function "$\cdot$" returning the empty list. This means that notations as structural features have no semantic consequence. They are just used by other processes, such as parsers and presenters. The validity function, checks that the pragmatic declarations are constants of the form $c = P^O \parallel E$ where any variables in E serve as notation arguments. They are used for the arity and can be referenced by realization specs in P^O. This definition is useful as it permits representing non-compositional notations involving several constants such as $sin^2(x)$ – as opposed to the compositional $(sin(x))^2$ – which involves both sin and $power$. But, the most common case are pragmatics of the form $c = P^O \parallel (s\ \vec{x})$. Then s is the symbol for which this notation applies and $\vec{x}$ are the arguments. In both cases, P^O is a narration which can make use of the arguments as needed, and presenters replace the arguments with the actual values when presenting applied instances of the symbol. Additional properties of the notation are optionally given using metadata. The possible attributes and values are given in Figure 6.2 and are inspired from OMDOC's notations. The same properties form the presentation context of IMMT realization specs and can be used to override options of the notation itself. For example, to give different precedence to an argument. Note that it is up to the IMMT exporters to actually make use of this information when presenting IMMT objects.

Example 16. Consider the definition:

(6.1) *"The special linear group of degree n over a field F, denoted $SL(n, F)$, is the set of $n \times n$ matrices with determinant 1, with the group operations of ordinary matrix multiplication and matrix inversion."*

From the perspective of IMMT the text above declares three things:

Name	Syntax	Intuition
type	`parse` | `pres` | `verb`	for parsing (text), presentation (html) or verbalization (e.g. english)
lang	`String`	default content type of this notation (e.g. `text`, `mathml` or `en`, `de` for verbalizations)
precedence	`Integer`	priority when parsing, bracket detection
variant	`String`	string identifier (name) of this notation

Figure 6.2: Notation Scope in IMMT

- A symbol for special linear group (will call it slg below for brevity) with a flexiformal definition.
- A binary presentation notation $SL(n, F)$ for the symbol slg.
- A binary verbalization notation "special linear group of degree n over a field F" for the symbol slg.

The corresponding representation in IMMT for each of the two notations above is a derived declaration of the feature $\mathfrak{Notation}$. Its pragmatics are a constant with its definition a term of the form $P^O \parallel (slg\ n\ F)$. For the presentation notation the narrative aspect P^O is a MathML expression with object-level realization specs (see Section 5.1.2) for the variables:

```
<mrow>
  <mo>SL</mo>
  <mo>(</mo>
  <obj-real-spec position="1">
    <pres-cont language="mathml"/>
  </obj-real-spec>
  <mo>,</mo>
  <obj-real-spec position="2">
    <pres-cont language="mathml"/>
  </obj-real-spec>
  <mo>)</mo>
</mrow>
```

with metadata

```
<meta format="mathml" variant="default"/>
```

For the verbalization notation, the corresponding narration is :

```
special linear group of degree
<obj-real-spec position="1">
  <pres-cont language="mathml" type="pres"/>
</obj-real-spec>
over a field
<obj-real-spec position="2">
  <pres-cont language="mathml" type="pres"/>
</obj-real-spec>
```

with metadata

```
1 <meta type="verb" format="text" variant="default"/>
```

Note, that the verbalization notation above, explicitly uses the `type="pres"` attribute for the arguments. This prioritizes (assuming a correctly implemented exporter) the math notations for its arguments (n and F in this case) over any available verbalization notations. However, the presentation context for the term being presented (see Section 5.1.2) can be used to override that preference. For instance, by choosing a concise verbalization for the set of real numbers rather than the math notation ($\mathbb{R}$) as in: "*Special linear group of degree n over the reals*".

The notations described as above are effectively OMDoc notations [Koh10] only expressed in terms of iMMT primitives. We discuss OMDoc notations further in Section 7.2 where we present our import of several sTeX libraries in iMMT. sTeX, as discussed in Section 4.1, is an expressive, LaTeX-based surface syntax for OMDoc.

6.1.2 Mathematical Structures

To represent mathematical structures we introduce the $\mathfrak{MathStruct}$ structural feature. Before defining $\mathfrak{MathStruct}$ we define a series of sub-features motivated by constructs that are used to define mathematical structures in CML.

In mathematical practice, a mathematical structure instance has a *name*, a set of *primitives* and a set of *properties* for those primitives. We further separate the primitives into set (or type) declarations (e.g. G for a group or (F, V) for vector space) and operator declarations (e.g. $\circ$ for monoid or $\sqcap$ for a meet-semilattice). We include nullary operators such as elements of the primitive sets (e.g, e monoid) in the operator list.

Therefore, we define three dedicated subfeatures of $\mathfrak{MathStruct}$ for each category discussed above:

First, we define an unary structural feature $\mathfrak{TypeDecl}$ which is used for declaring the primitive types (usually sets in CML) used to form a mathematical structure. The argument E represents the type of the resulting primitive and is typically set. The validity predicate requires that there are no pragmatic declarations (only term arguments).

Second, we define an unary structural feature $\mathfrak{OpDecl}$ which is used for declaring operators on types of the mathematical structure. The argument E represents the type of the resulting operator the validity constraints require that it uses the types declared as primitive types in its construction.

Third, we define an unary structural feature $\mathfrak{PropDecl}$ for declaring mathematical properties of the types and operators forming the mathematical structure. The argument E represents the assertion that must hold and the validity constraints require that it is a proposition.

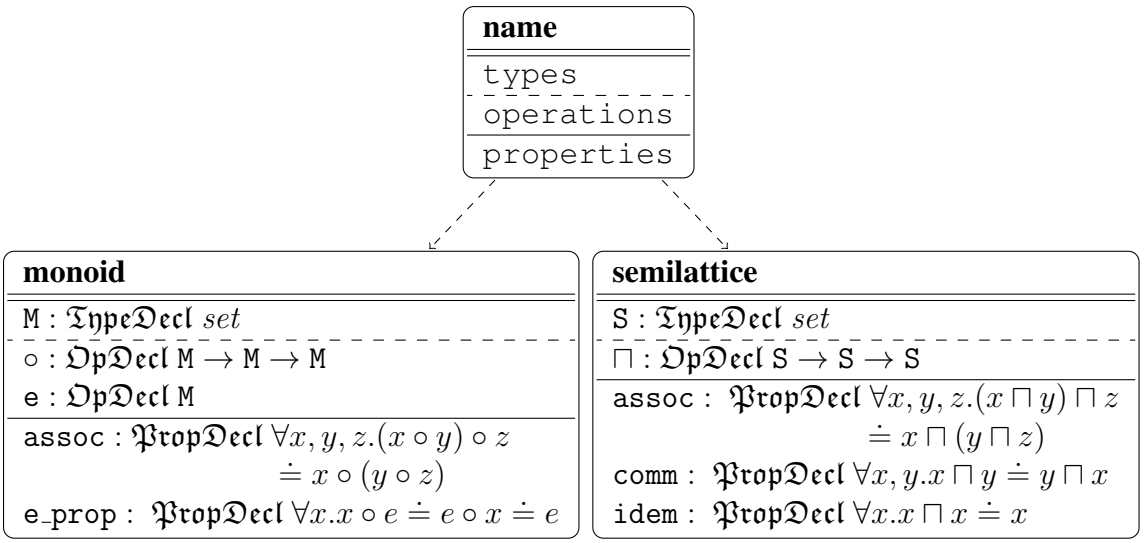


Figure 6.3: Example instances of the $\mathfrak{MathStruct}$ feature

Figure 6.3 shows two examples of mathematical structures that can be declared in a straightforward way using the subfeatures introduced above. However, as observed in Section 2.1, in practice mathematical structures are not always defined directly by their components but, instead, extend or instantiate other pre-existing structures. Therefore, in addition to the three basic subfeature described above, we declare two additional subfeatures for representing commonly occurring definitional mechanisms observed in practice.

First, we define a unary subfeature $\mathfrak{Extend}$ where the argument E is a reference to the mathematical structure being extended. The validity constraints require that the reference is valid.

Second, we define a binary subfeature $\mathfrak{Inst}$ where the first argument E_1 is a reference to the mathematical structure being instantiated and the second argument Γ_2 is a list of local names to be used in the current $\mathfrak{MathStruct}$ for the primitives of the imported one. The validity constraints check, in addition to the validity of the reference, that the length of the list of local names matches the number of primitives of the referenced mathematical structure.

Now, we can define $\mathfrak{MathStruct}$ as a nullary structural feature. The validity function requires that each of its pragmatic declarations are instances of one of the subfeatures declared above. Additionally, it checks that there is at most one instance of the $\mathfrak{Extend}$ feature. The elaboration function $\Theta \vdash_T ms = \mathfrak{MathStruct}\ E\ \{\Sigma\} \rightsquigarrow \{\Sigma'\}$ is defined for each statement $S \in \Sigma$ as follows:

- if S is $s = \mathfrak{TypeDecl}\ E\ \{\cdot\}$ or $s = \mathfrak{OpDecl}\ E\ \{\cdot\}$ then we add $s : E$ to Σ'
- if S is $s = \mathfrak{PropDecl}\ E\ \{\cdot\}$ then we add $s : \Vdash E$ to Σ'
- if S is $s = \mathfrak{Extend}\ \mathtt{ms}_2\ \{\cdot\}$ then we add $\mathtt{include}\ \mathtt{ms}_2$ to Σ'
- if S is $s = \mathfrak{Inst}\ \mathtt{ms}_2\ \vec{l}\ \{\cdot\}$ then:

$\Sigma' = \{$
include $g?monoid$
`invertibility` : $\Vdash \forall x \in G.\exists y \in G.x \circ y = e$
$\}$

Figure 6.5: Elaboration of the *group* $\mathfrak{MathStruct}$

- for each local name l_i we find the corresponding (by order) primitive declaration $s_i : E_i$ in $\mathtt{ms2}$
- we construct the partial translation function t such that $t(s_i) = l_i$ for all i
- if l_i is new (not already defined) in Σ' we add $l_i : \bar{t}(E_i)$ to Σ'
- otherwise, we check that the type of the declaration $l_i : E'$ from Σ' is compatible with (i.e. equal to) $\bar{t}(E_i)$
- finally, for every property $p_j : \Vdash E_j$ from $\mathtt{ms}_2$ we add $p_j : \Vdash \bar{t}(E_j)$ to Σ'

The intuition of the elaboration algorithm for $\mathfrak{Inst}$ is that we copy over the declarations from the target, but while renaming appropriately to the given names. Moreover, we skip over copying declarations if the name exists and the types match (which should be checked by the validity predicate). Then, translating over the terms ensures that the references are appropriately updated so that well-formedness is preserved. The algorithm is demonstrated in Example 18.

Below we look at two examples of mathematical structures from Section 2.1 and how they can concretely be represented in iMMT using the $\mathfrak{MathStruct}$ feature.

Example 17. Consider the definition of *group* from Example 2.28 which we repeat on the left side of Figure 6.3. On the right side, we show a definition of *group* as a $\mathfrak{MathStruct}$ that follows the structure of the narrative definition. Concretely, it extends the pre-defined $\mathfrak{MathStruct}$ `monoid` with one additional property, namely that every element has an inverse. We assume `monoid` is defined in the natural way, with G as its base set, $\circ$ its operation, and e its neutral element.

A **group** is a monoid (§**2**, no. 1, Definition 1) in which every element is invertible.

`group` : $\mathfrak{MathStruct} = \{$
_ : $\mathfrak{Extend}$ $g?\mathtt{monoid} = \{\cdot\}$
$invertibility$: $\mathfrak{PropDecl}$ $\forall x \in G.\exists y \in G.x \circ y = e$
$\}$

Figure 6.4: Defining *group* as a $\mathfrak{MathStruct}$ in iMMT

Then, the elaboration follows the algorithm described above. We first elaborate the $\mathfrak{Extend}$ instance to an include of `monoid`. Then, we add the elaboration of the $invertibility$ $\mathfrak{PropDecl}$. The resulting set of declarations Σ' is shown in Figure 6.5.

Example 18. Consider the definition of *ring* from Example 2.29 which we repeat on the left side of Figure 6.6. On the right side, we show its representation as a $\mathfrak{MathStruct}$ that mirrors the structure of the narrative version.

$$
\begin{aligned}
&\Sigma' = \{\\
&\quad R : \mathit{set}\\
&\quad + : R \to R \to R\\
&\quad 0 : R\\
&\quad - : R \to R\\
&\quad * : R \to R \to R\\
&\quad 1 : R\\
&\quad \texttt{comgroup/assoc} : \Vdash \forall x, y, z.(x + y) + z = x + (y + z)\\
&\quad \vdots\\
&\}
\end{aligned}
$$

Figure 6.7: Elaboration of the *ring* $\mathfrak{MathStruct}$

Definition 4. Let $\langle R, +, 0, - \rangle$ be a commutative group, and $\langle R, *, 1 \rangle$ a monoid, such that $\langle R, +, * \rangle$ is a ringoid, then we call $\langle R, +, 0, -, *, 1 \rangle$ a **ring**.

$$
\begin{aligned}
&\texttt{ring} : \mathfrak{MathStruct} = \{\\
&\_ : \mathfrak{Inst}\ \texttt{commgroup}\ R, +, 0, - = \{\cdot\}\\
&\_ : \mathfrak{Inst}\ \texttt{monoid}\ R, *, 1 = \{\cdot\}\\
&\_ : \mathfrak{Inst}\ \texttt{ringoid}\ R, +, * = \{\cdot\}\\
&\}
\end{aligned}
$$

Figure 6.6: Defining *ring* as a $\mathfrak{MathStruct}$ in IMMT

The resulting derived declaration contains three $\mathfrak{Inst}$ sub-features one for each instantiation of another mathematical structure used in the original definition. The elaboration result Σ' is constructed using the elaboration algorithm described above and is shown in Figure 6.7. First the primitive declarations for the base set R and the $+$, 0 and $-$ operations are added to the signature by the $\mathfrak{Inst}$ of `comgroup`. For the second $\mathfrak{Inst}$ (of `monoid`), R already exists and is matched since its type (set) is compatible. Therefore, afterwards, $*$ and 1 are added with their type now referring to the same R as $+, 0$ and $-$ due to the translation function. Then, for the last $\mathfrak{Inst}$ of `ringoid` all local names R, $+$ and $*$ are matched with the existing ones. Finally, the properties, (associativity, commutativity, distributivity, etc.) are added modulo the translation function t that updates the referenced names accordingly. The names of the properties are namespaced using the "/" separator to avoid name clashes.

6.1.3 Structured Proofs

Proofs are an essential concept in mathematics and related fields. Strict MMT takes the logical view of proofs as a tree of applications of inferences rules, but the notion of proof is much wider in mathematical practice. Proofs often appear as statement or module-level elements with rich internal structure containing inline definitions, lemmas, sub-proofs, etc. as proof steps. To explore how such structured proofs can be modeled with structural features we represent OMDOC proofs (which are quite generic and expressive in their own right) as a structural feature

with an elaboration based on our informal foundation IF. The intuition of the elaboration is that it merges the proof steps using FOL connectives into one strict MMT term representing the proof as an application tree. Therefore, the elaboration result is a single MMT constant declaration with its definition being the proof term.

The OMDOC reasoning steps used to construct proofs are *symbol declarations*, *definitions*, *hypotheses* or *derivations*. Therefore, before defining proofs, we introduce a series of subfeatures:

- a feature $\langle \mathfrak{OMDecl}, \mathit{IF}, 1, \mathcal{V}_{\mathfrak{OMDecl}}, \mathcal{E}_{\mathfrak{OMDecl}} \rangle$ corresponding to OMDOC symbol declarations, where the argument represents the type of the symbol.
- a feature $\langle \mathfrak{OMDef}, \mathit{IF}, 2, \mathcal{V}_{\mathfrak{OMDef}}, \mathcal{E}_{\mathfrak{OMDef}} \rangle$ corresponding to OMDOC symbol definitions, where the two arguments represent the type and respectively, definition of the symbol.
- a feature $\langle \mathfrak{OMHyp}, \mathit{IF}, 1, \mathcal{V}_{\mathfrak{OMHyp}}, \mathcal{E}_{\mathfrak{OMHyp}} \rangle$ corresponding to OMDOC hypotheses, where the argument represents the proposition being assumed to hold.
- a feature $\langle \mathfrak{OMDer}, \mathit{IF}, 2, \mathcal{V}_{\mathfrak{OMDer}}, \mathcal{E}_{\mathfrak{OMDer}} \rangle$ corresponding to OMDOC derivations, where the two arguments represents the proposition being derived and, respectively, its proof.

In all cases above, the validity predicate requires that the list pragmatic declarations is empty. The elaboration functions are described as part of the elaboration proofs discussed below.

An OMDOC *structured proof* is a structural feature $\langle \mathfrak{Proof}, \mathit{IF}, 1, \mathcal{V}_{\mathfrak{Proof}}, \mathcal{E}_{\mathfrak{Proof}} \rangle$ where the argument is the proposition being proved and the pragmatics are MMT statements representing the proof steps. The validity predicate ensures that the pragmatics are instances of proof steps, meaning either $\mathfrak{OMDecl}$, $\mathfrak{OMDef}$, $\mathfrak{OMHyp}$, $\mathfrak{OMDer}$ or of $\mathfrak{Proof}$ itself. The last one ensures that, like in OMDOC, proofs can nest and be used as reasoning steps to construct larger proofs.

The elaboration function $\Theta \vdash_T p = \mathfrak{Proof}\ E_1\ \{\Sigma\} \rightsquigarrow \{\Sigma'\}$ produces a single constant $p : \Vdash E_1 = E_2$ where E_2 (the proof) is produced sequentially based on each declaration in Σ as follows:

- for an instance $x = \mathfrak{OMDecl}\ E\ \{\cdot\}$ we produce $\forall_I(\lambda\ x : E.E')$ where E' is the elaboration of the later statements in Σ and $\forall_I$ is the $\forall$ introduction rule in IF.
- for an instance $h = \mathfrak{OMHyp}\ E\ \{\cdot\}$ we produce $\Rightarrow_I(\lambda\ h : E.E')$ where E' is the elaboration of the later statements in Σ and $\Rightarrow_I$ is the implication introduction rule in IF.
- for an instance $c = \mathfrak{OMDef}\ E_a, E_b\ \{\cdot\}$ we produce $\bar{t}(E')$ where E' is the elaboration of the later statements in Σ and t is the function replacing c with E_b.
- for an instance $d = \mathfrak{OMDer}\ E_a, E_b\ \{\cdot\}$ we have two cases:
  - if this is the last statement in Σ then we produce E_b
  - else, we produce $\bar{t}(E')$ where E' is the elaboration of the later statements in Σ and t is the function replacing d with E_b.

- finally, for an instance $sp = \mathfrak{Proof}\ E_a\ \{\Sigma_{sp}\}$ we elaborate recursively to obtain its proof term E_{sp} then treat that exactly like for the derivation step above:
  - if this is the last statement in Σ then we produce E_{sp}
  - else, we produce $\overline{t}(E')$ where E' is the elaboration of the later statements in Σ and t is the function replacing sp with E_{sp}.

The intuition of the elaboration algorithm above is that the linear proof encoded as pragmatic statements is elaborated to a tree structured IMMT term but preserving the intended semantics. Undefined symbols become universally quantified, hypothesis become assumptions for proving implications and defined symbols are, where applicable, replaced with their definition.

Note that alternative elaborations of such proofs can be envisioned depending on the logical meta-theory where the $\mathfrak{Proof}$ structural feature is declared. For instance, [ASC06] shows a formal correspondence between OMDOC proofs and the $\overline{\lambda}\mu\tilde{\mu}$-calculus which can be straightforwardly re-framed as an elaboration function into a particularly expressive meta-logic (i.e. $\overline{\lambda}\mu\tilde{\mu}$).

6.1.4 Dynamic Theories

Acknowledgement. *Part of the results in this Section are based on collaborative work with Michael Kohlhase have been published in [IK15b].*

Strict IMMT (same as MMT) already provides theories as reusable containers of mathematical statements. However, this is insufficient for informal mathematics where, often, such containers are not declared but inferred and aggregated from multiple sources. In practice, mathematicians often abstract away details of an underlying development and focus on the useful symbols and theorems that implicitly form the mathematical theory. Basic examples are the different ways to define natural or real numbers, groups or topologies.

We observed this as the phenomena **P6b** Dynamic Theories in Section 2.1. Additionally, a very similar concept is discussed in [CF08] as *high-level theories*, in analogy with high-level programming. Moreover, [CFK14] introduces *realms* as a theory-graph construct for representing such dynamic mathematical modules in the formal world.

In this Section, we build on this existing work, but re-frame it in terms of IMMT structural features as well as extend it to cover the phenomena as observed in Section 2.1. But, first, we briefly introduce realms and refer to [CFK14] for details.

Realms Intuitively, a realm is a set of modules in a theory graph Θ (i.e. a subgraph of Θ) that abstracts from the development and provides practitioners with the useful symbols and theorems via an *interface theory*.

An important concept for realms is that of a *conservative extension* which usually occurs when a theory includes another and contains only theorems and derived symbols (i.e. adds no axioms or

primitive symbols). An essential property of conservative extensions is that if S' is a conservative extension of S then there is view v between T and S iff there is a view between T and S' in the same direction. In fact, we will often talk about views *modulo conservativity* below.

Figure 6.8 shows a prototypical realm with F as its interface theory (also called a *face*) and n pillars each representing a different (yet isomorphic) development of the concepts in the face. Each pillar is a conservative development in the sense that all theories in a pillar are conservative extensions of a bottom theory (denoted with $\bot$). A top theory (denoted with $\top$) aggregates all symbols, axioms and theorems declared within the pillar. The view pairs at the bottom establish the equivalence of the pillars and the n views I_k capture the relation of interface-implementation between the face and each pillar.

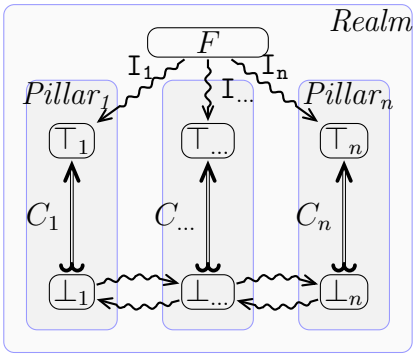


Figure 6.8: The Architecture of a Realm

Theory Isomorphisms as a Feature To define realms as structural features we first define a feature for theory isomorphisms, which are the relations between the pillars of a realm. But theory isomorphisms are useful also on their own. Different definitions or developments yielding equivalent concepts is a regular occurrence in both formal and informal mathematics. In module systems like MMT or IMMT, this typically manifests as theories that are *isomorphic*. We introduce theory $\mathfrak{Isomorphism}$ as a binary structural feature for IF. Its two arguments are just references (global names) to the theories in question, and its pragmatics are pairs of equivalent constants that define the isomorphism (represented as MMT constants). Note that the general case where the isomorphism can map constants in a theory to *objects* in the other is also possible but we omit the details here for brevity.

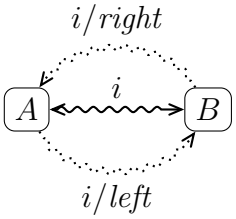


Figure 6.9: Elaboration of Theory Isomorphisms

We define the elaboration of a derived module $i : IF = \mathfrak{Isomorphism}\ A, B\ \{\Sigma\}$ to be the two views $i/left : A \to B = \{\Sigma\}$ and $i/right : B \to A = \{\Sigma_r\}$ where Σ_r is constructed inductively as follows:

- Σ_r starts out as the empty body $\{\cdot\}$
- for every constant $c_a = c_b$ in Σ (representing a symbol pair) we add $c_b = c_a$ to Σ_r

Effectively, Σ_r is the reverse of Σ and we obtain the diagram in Figure 6.9 as the result of elaboration.

Realms as a Structural Feature Now we are ready to reframe realms as a structural feature. We first define pillars as an auxiliary structural feature $\mathfrak{Pillar}$ with flexible arity. The arguments

represent (references to) the theories that form the pillar with the first argument being the base theory.

Then, a realm is a nullary structural feature $\mathfrak{Realm}$ whose validity predicate requires that the pragmatic declarations of its instances are either pillars or isomorphisms. The elaboration function for a realm $r : IF = \mathfrak{Realm}\ \{\Sigma\}$ with n pillars produces a theory $r/face$ (the interface theory) together with n views r/v_i that justify the interface theory as an abstraction of each pillar. The theory and views are constructed following the *face-generation* algorithm described below which basically traverses the pillars to produce the face by aggregating statements that are unique modulo the given isomorphisms. Figure 6.10 shows the diagram obtained as the result of elaboration.

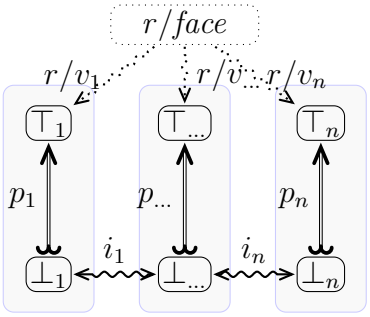


Figure 6.10: Elaboration of a Realms

Generating Realm Faces We introduced the algorithm for generating the faces of realms as induced theories in [IK15b]. Effectively, we consider their face as inducible from the pillars and discuss below how we mechanize that. Generally, the face is produced from the statement in the pillars modulo the equivalence views and there are two main difficulties.

The former is abstracting away (i.e. removing from the face) those statements that are only needed for development in the pillars. For example defining functions using sets might require defining pairs first but the face should only contain functions and their properties.

Therefore we introduce a structural feature $\mathfrak{Private}$ that contains a single statement S and elaborates to it – so it does not change the strict perspective of theories where it is used. The purpose of $\mathfrak{Private}$ is to allow pillar developers to mark statements that are not meant to be exported into the face by wrapping them in a $\mathfrak{Private}$ derived declaration. Then, for the purpose of the face generation algorithm discusses below we can safely ignore all instances of the feature $\mathfrak{Private}$ to produce the face as intended by the pillar developers.

The latter is generating unique names for the statements. The algorithm we give is simply an heuristic for generating those names in some cases. Otherwise, we default to using the name of each pillar as a prefix (see renaming in Definition 14 above) to ensure uniqueness whenever the heuristic fails. Therefore, alternative and perhaps better algorithm could be designed in the future to improve on the work described here. Note that, from a category theory perspective this generally amounts to generating a co-cone of the pillars. It is not always a co-limit because pillars can, and usually do, abstract away implementation details from the pillars. Even so, co-limits are unique only up to isomorphism and, in practice, the particular choice of names for the face does matter. Moreover, there is no general, optimum solution for choosing the names (see [CMR16] for a discussion) which is why the algorithm we give is merely a reasonable heuristic for our particular case.

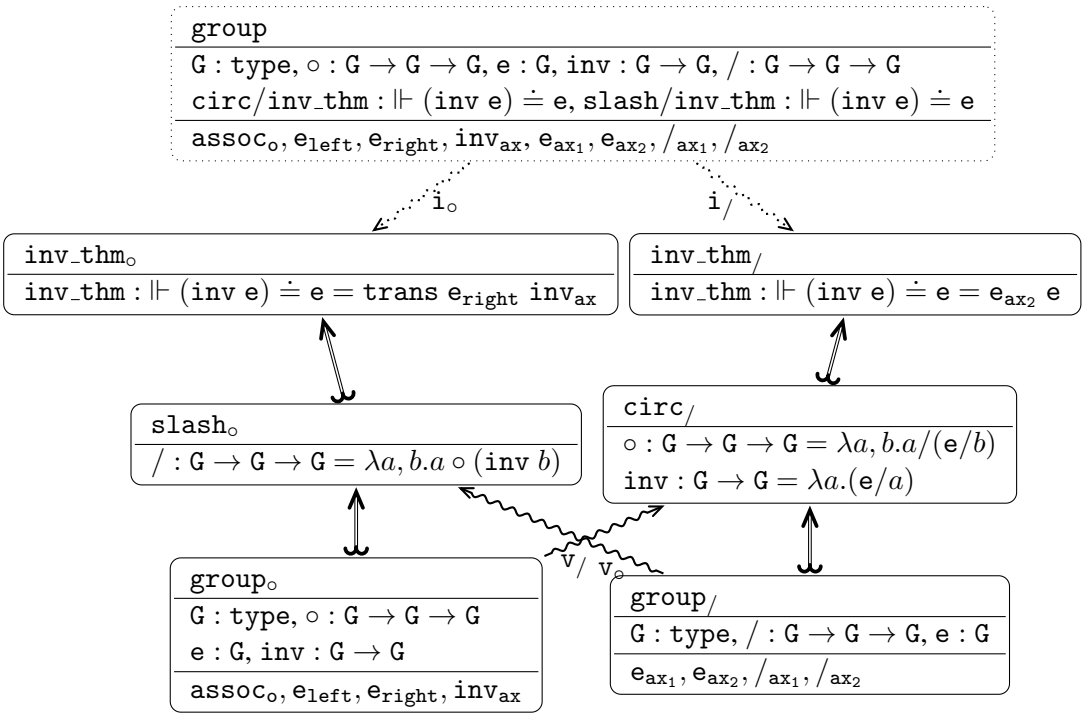


Figure 6.11: A Realm of Groups

For the generated face, the URI g of the theory with the realm specification induces the IMMT URI of the face, and the IMMT URIs of the symbols inside are induced from the pillar names. To generate the face we need to solve three main issues:

1. select which symbols from each pillar should be in the face
2. merge equivalent symbols (such as the e in Figure 6.11)
3. resolve naming conflicts (equivalent symbols with different names and distinguishable symbols with same name)

Before we formalize this in Definition 21 below, let us adapt the groups realm from [CFK14] to the IMMT setting.

Example 19. Figure 6.11 shows a realm with two pillars for the isomorphic group definitions based on composition $\circ$ and, respectively, division $/$ in the usual way. The corresponding realm specification is in Figure 6.12. We have added the unit e with its axiom $\mathtt{e}_{\mathtt{ax}_1}, \mathtt{e}_{\mathtt{ax}_2}$ to $\mathtt{group}_/$ even though it is mathematically redundant – can be defined as x/x for all x in G – because that allows us to show all aspects of the face generation algorithm below.

Theories $\mathtt{slash}_\circ$ and $\mathtt{inv\_thm}_\circ$ as well as $\mathtt{circ}_/$ and $\mathtt{inv\_thm}_/$ are each conservative developments of $\mathtt{group}_\circ$ and $\mathtt{group}_/$ respectively as they only introduce defined symbols. The views $\mathtt{v}_\circ$ and $\mathtt{v}_/$ ensure the equivalence of the two pillars with the obvious assignments. The proofs that the axioms hold (i.e. the assignments for them in the views) are all straightforward and omitted for simplicity. The *face* for the realm is shown in theory group and has the intuitive shape, containing all the important concepts as primitive symbols and all their properties as axioms. We

$$
\begin{array}{l}
\texttt{group} = \mathfrak{Realm}\ \{ \\
\quad \texttt{circ} = \mathfrak{Pillar}\ \texttt{group}_\circ,\ \texttt{slash}_\circ\ \{\cdot\} \\
\quad \texttt{slash} = \mathfrak{Pillar}\ \texttt{group}_/,\ \texttt{circ}_/\ \{\cdot\} \\
\quad \texttt{v} = \mathfrak{Isomorphism}\ \texttt{circ},\ \texttt{slash}\ \{\texttt{v}_\circ,\ \texttt{v}_/\} \\
\}
\end{array}
$$

Figure 6.12: The realm of groups as a derived declaration

explain how the face was produced below.

Definition 20. We define the following partial ordering on symbols in a realm $\texttt{r}$. Let $\texttt{t}$ and $\texttt{s}$ be symbols in theories $\texttt{T}$ and $\texttt{S}$ respectively such $\texttt{T}$ and $\texttt{S}$ are in different pillars. If there is an assignment $\texttt{s} = \texttt{t}$ in one of the pillar equivalence views then we write $\texttt{s} \leq_\texttt{r} \texttt{t}$. If there is also an assignment $\texttt{t} = \texttt{s}$ in such a view then we write $\texttt{s} =_\texttt{r} \texttt{t}$, otherwise $\texttt{s} <_\texttt{r} \texttt{t}$. We call a symbol $\texttt{s}$ **essential** in $\texttt{r}$ if there is no symbol $\texttt{t}$ such that $\texttt{s} <_\texttt{r} \texttt{t}$.

The intuition behind Definition 20 is that if there is an assignment from $\texttt{s}$ to $\texttt{t}$ (or conversely $\texttt{t}$ to $\texttt{s}$, or both) then $\texttt{s}$ and $\texttt{t}$ represent the same concept at the realm level. So, if possible, we can add only one of the two constants in the face to represent that concept. Then, the ordering captures the fact that $\texttt{s}$ is a primitive concept in its realm while $\texttt{t}$ is derived, possibly only to give the equivalence view. For example, in Figure 6.11 the symbols $\texttt{inv}$ and $\circ$ from theory $\texttt{circ}_/$ are derived.

Definition 21. Let $\texttt{r}$ be a realm as defined above. We generate a face for $\texttt{r}$ by adding copies of all essential symbols with their type but not their definition (if any) – following the specification of a realm face from [CFK14] – with the following provisions:

1. If two (or more) essential symbols in different pillars have the same name we prefix all of them with the pillar name (to ensure unique names)
2. If n essential symbols are equal (with respect to $=_\texttt{r}$) we first order them $\texttt{s}_1, \ldots, \texttt{s}_n$ (based on the order of their respective pillars within the pragmatics of r) and then:
   - add $\texttt{s}_1$ as normal
   - for every $\texttt{s}_i$, if it has the same name as $\texttt{s}_1$ we skip it (effectively merge the two statements)
   - otherwise, we add the constant $\texttt{s}_i = \texttt{s}_1$ (effectively use $\texttt{s}_i$ as an alias of $\texttt{s}_1$)

Example 22. The face for $\texttt{group}$ from Figure 6.11 is generated following the algorithm from Definition 21. The essential symbols (omitting axioms for simplicity) are $\texttt{G}, \circ, \texttt{e}, \texttt{inv}, \texttt{inv\_thm}$ for the first pillar and $\texttt{G}, /, \texttt{e}, \texttt{inv\_thm}$ for the second. We have two name clashing pairs: $\texttt{group}_\circ?\texttt{e}$ and $\texttt{group}_/?\texttt{e}$ as well as $\texttt{inv\_thm}_\circ?\texttt{inv\_thm}$ and $\texttt{inv\_thm}_/?\texttt{inv\_thm}$. For the first pair ($\texttt{e}$) we have an equality since $\texttt{v}_\circ$ and $\texttt{v}_/$ assign them to each other so we merge. Then, for the second

pair (`inv_thm`) there is no assignment so we cannot merge. Therefore, we follow case 1. of the algorithm and prefix them with the pillar name producing `circ/inv_thm` and `slash/inv_thm` to obtain the face shown in Figure 6.11.

Curating Realms through Alignments As already mentioned above, the face generation (elaboration) algorithm we described is just a heuristic and does not always produce the intuitive result. For instance, the intuition is that it would merge equivalent constants in different pillars. But, occasionally, constants that are essentially equivalent will appear as different because of slightly different formalizations. A common case is identical theorems but with different proofs as is the case of `inv_thm` in Example 22.

In [CFK14] realm faces are meant to be manually generated and curated to avoid such issues. However, we propose an alternative method of curating faces by giving *alignments* between pillar theories to establish symbols as being equivalent. This idea is inspired from [KRSC11], and is further discussed in [KKMR16] but in the context of equivalent concepts across formal libraries.

Definition 23 (Alignment)**.** An **alignment** is a view pair $v_1 : \hat{S} \to T$ and $v_2 : \hat{T} \to S$ where $\hat{S}$ is is a copy of S but which omits all definitions in S. We call $\hat{S}$ the **abstraction** of S.

The abstraction operation $\hat{\cdot}$ is needed to allow us to assign a new interpretation for defined symbols which would otherwise be translated automatically by iMMT/MMT via definition expansion. We will concentrate on the case where S and T are in different pillars here. Note that, in [KRSC11], the idea of generalizing MMT views to allow giving assignments for already defined constants was proposed. This is not yet realized in the implementation but would allow us to skip the abstraction step and just give the views directly.

In iMMT we can represent alignments using a binary structural feature $\mathfrak{Alignment}$ where the two arguments are the global names of the theories. Then, the validity predicate checks that its pragmatics are a pair of views as described above. We add it as a subfeature of $\mathfrak{Realm}$ and enhance the elaboration function to make use of it as described above.

For instance, take the situation in Figure 6.11 where the (trivial) theorem `inv_thm` proving that `e` is its own inverse appears in each pillar. Still, the proofs are different over the translation so that both symbols appear different at the iMMT level. However, we can fix the problem by giving an alignment between the two theories containing the theorem. Listing 6.13 below shows the alignment and the resulting, curated face.

The central idea here is that the algorithmic procedure for generating the face does not preclude manual curation. On the contrary, by adding these knowledge items (such as alignments) it makes the process more transparent and systematic. Therefore, it potentially allows other, future applications to make use of this information as it is now explicitly present in the diagram.

$$\texttt{inv\_thm} = \mathfrak{Alignment}\ \texttt{circ},\ \texttt{slash}\ \{$$
$$\quad \text{view } \texttt{a}_{/} : \texttt{inv\_}\hat{\texttt{thm}}_{\circ} \to \texttt{inv\_thm}_{/} = \{\texttt{inv\_thm} := \texttt{inv\_thm}\}$$
$$\quad \text{view } \texttt{a}_{\circ} : \texttt{inv\_}\hat{\texttt{thm}}_{/} \to \texttt{inv\_thm}_{\circ} = \{\texttt{inv\_thm} := \texttt{inv\_thm}\}$$
$$\}$$
$$\text{theory } \texttt{group} = \{\texttt{G} : \texttt{type}, \texttt{e} : \texttt{G}, \ldots, / : \texttt{G} \to \texttt{G} \to \texttt{G} \ldots, \texttt{inv\_thm} : \Vdash (\texttt{inv}\ \texttt{e}) \doteq \texttt{e}\}$$

Figure 6.13: Alignment Example

6.1.5 Recaps

Acknowledgement. *The results in this section are collaborative work with Michael Kohlhase and have been published in [IK15a].*

In this Section we look more closely at the examples for recaps from Section 2.1 and how each can be represented using theory graphs and structural features in IMMT.

First, we look at the aspects common to all papers we analyzed to form an intuition of the theory graphs structures that are needed. In every example, recaps aggregate definitions and theorems from similar developments to establish a common ground. The result is typically referred to as "the literature" and is used in two ways.

First, the paper uses the literature as the basis upon which the results of the paper are built. We call this process *aggregation*, where the knowledge from the various sources is integrated, with the irrelevant details abstracted away. Then, the paper makes use of definitions, theorems and notations from the literature to produce its result.

Second, the results from the paper implicitly contribute back to the literature and can be used in further papers within that field. We call this process *dissemination* which, in mathematical practice, happens implicitly when the paper is published.

In this thesis we claim that the processes described above can be adequately modeled in IMMT based on realms as defined above. The central idea is that we represent the literature as a realm with the aggregation that the paper makes use of as the face of the realm. Then, dissemination is modeled as a process of *extending* the realm with the results from the current paper. The basic theory behind extending realms was already introduced in [CFK14], and we refer to that for details. Here we just apply it to modeling dissemination as part of representing recaps in IMMT.

Realms as a Model for Dissemination & Aggregation Figure 6.14 shows the typical case for the representation of a paper as part of a theory graph. The "literature" for the mathematical theory to which the paper contributes is represented as a realm with a face and several pillars. The paper references a document within the field, that is naturally part of a pillar and grounds the recap theory. The contribution of the paper is a theory in itself that includes the recap theory and is a conservative extension of it. Again, the fact that we are representing the contribution in a single theory is a simplification for presentational simplicity which does not lead to a loss of

generality. The view v ensures that the paper can make use of concepts and theorems from the realm, as they can be accessed via v.

In our analysis we first restrict ourselves to the case where there is a single recap for simplicity and expositional clarity. This already covers the majority of research papers we have analyzed; they mainly build on one earlier paper and extend it. Indeed, three of the four examples sources from Section 2.1 fall into this category, they import the definitions and terminology from a central cited paper, but call on others from the same realm for results, context, and support.

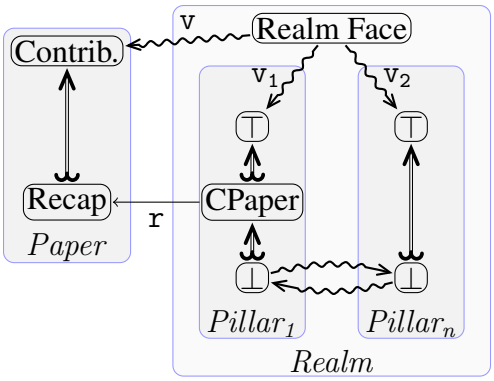
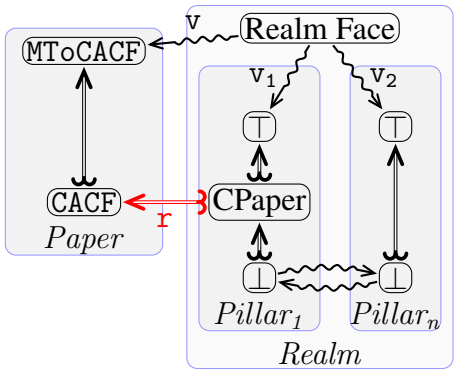


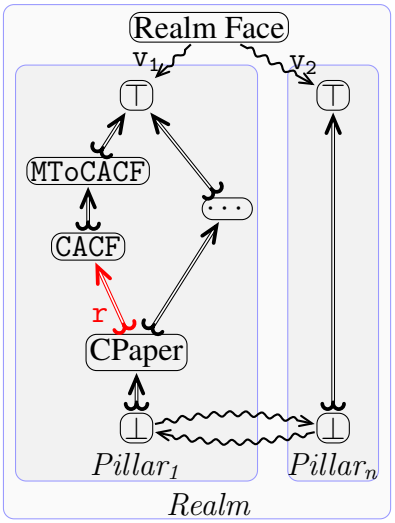
Figure 6.14: General Case for Recaps

We recognize four special cases for (single) recaps based on the nature of r and discuss each individually below. First we have to decide the home theory of the symbols that the recap introduces. If the home is the cited theory then r is an import and we have a *plain recap*. Otherwise, we have new symbols in the recap theory that are somehow related with the ones in the cited one. In that situation we have two sub-cases depending on the relation between the recap and cited theory: *equivalence recap* and *specialization recap*. Finally, we have the case where the paper builds on several others and, therefore, has *multiple recaps*.

Special case: Plain Recaps One situation is that of plain recaps where the relation r is an inclusion into the recap from the cited paper. Typically the include r is a conservative extension of the cited paper. For instance the "covers of the multiplicative group" from Example 2.39 directly uses the concept from the cited paper (CPaper), but gives a concise verbalization of its definition. This allows it to make use of the results in two other papers higher up in the pillar of the cited paper. The situation is shown in Figure 6.15a. Note that, if r is conservative, then we have a *pillar extension* for the realm which justifies the new paper becoming part of the realm's literature (see Figure 6.15b). It also makes v exist as induced by v_1 modulo conservativity.



(a) Publication Graph



(b) Aggregation Graph

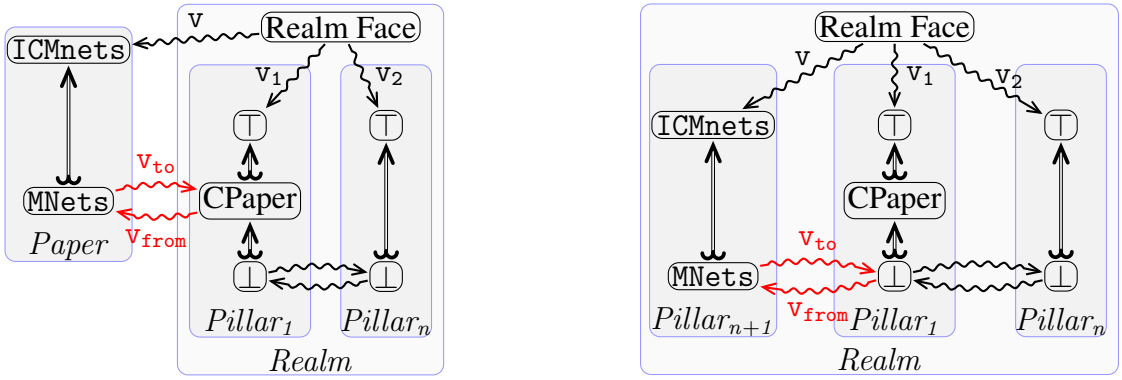
Figure 6.15: Plain Recaps (Example 2.39)

Plain recaps can also model the formal examples (e.g. Example 3) but in that situation it is not too interesting as we have the degenerate case for the realm itself.

Special Case: Equivalence Recap Another common situation is that of equivalence recaps where the relation $\mathtt{r}$ is an equivalence (isomorphism) between the two theories. We can represent the relation $\mathtt{r}$, in this case, as two views $\mathtt{v}_{\mathtt{to}}$ and $\mathtt{v}_{\mathtt{from}}$, one in each direction between the recap and the cited paper that ensure their isomorphism. Then, the view $\mathtt{v}$ is induced by $\mathtt{v}_{\mathtt{from}} \circ \mathtt{v}_1$ modulo conservativity. Moreover, the contribution of the paper carries over to the realm via the view $\mathtt{v}_{\mathtt{to}}$.

This occurs, for instance, in Example 2.40 where this intuition is explicitly written down in the paper as "There are several equivalent ways to define multinets." (although not proved). In fact it is the most common situation in the sample papers we studied.

Note that adding an equivalent definition corresponds to a *realm extension*, where the face is fixed, and the view from the face to the current theory can be postulated. Therefore, in Figure 6.16a the paper effectively extends the realm (or the current pillar) as introduced in Section 6.1.4. This corresponds to the mathematical practice of "contributing to" a field (or mathematical theory). This resulting realm after knowledge aggregation is shown in Figure 6.16b, where the new paper contributes a new pillar to the realm. The equivalence is ensured by $\mathtt{v}_{\mathtt{from}}$ and $\mathtt{v}_{\mathtt{to}}$ as we take into account conservativity to reduce them to the $\bot$ theory.



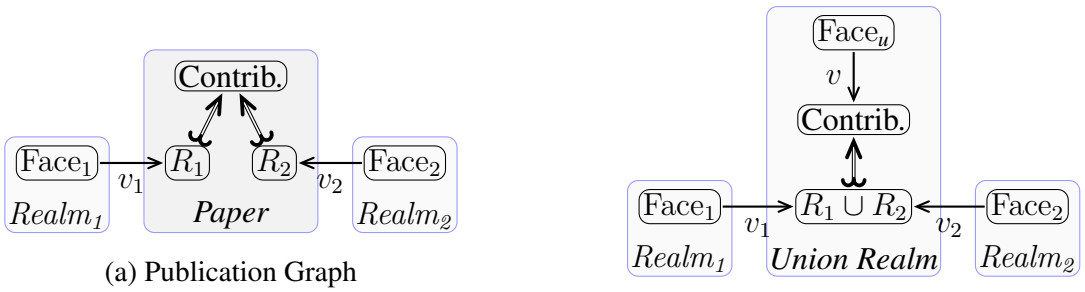
(a) Publication Graph

(b) Aggregation Graph

Figure 6.16: Equivalence Recaps (Example 2.40)

Special Case: Specialization Recap Thirdly, we have the case where $\mathtt{r}$ is a specialization relation that can be represented as a view $\mathtt{v_{from}}$ from the cited theory to the recap. Same as in the previous case, this ensures the existence of $\mathtt{v}$ as $\mathtt{v_{from}} \circ \mathtt{v_1}$ modulo conservativity. However it does not directly contribute the results of the paper back to the (same) realm as they concern only a special case of the concepts in the realm.

This is the case in Example 2.41 where the definition from the paper is a specialization of the one in the literature. In [CS09], the definition of the accelerated Turing machine involves a concrete step size (2^{-n}), whereas the definition it recaps allows arbitrary sequences of step sizes as long as their sum remains finite. Thus we have the situation in Figure 6.17. There is a view from $\mathtt{atm}$ to $\mathtt{ATM}$ that fixes the step size to 2^{-n}, but there cannot be a reverse view from $\mathtt{ATM}$ to $\mathtt{atm}$ as that is more general. However, we do have a view to $\mathtt{atm}(2^{-\mathtt{n}})$, which naturally arises in treatments of accelerated Turing machines as an example. That special case can form a realm of its own, namely the realm of accelerated Turing machines with step size 2^n. Then we can talk about aggregation with that realm (via the view $\mathtt{v_{to}}$) but we omit that here for simplicity – the aggregation is similar as for equivalence recaps, except with the specialization realm.



(a) Publication Graph

(b) Aggregation Graph

Figure 6.18: Multiple Recaps (Examples 2.42 and 2.43)

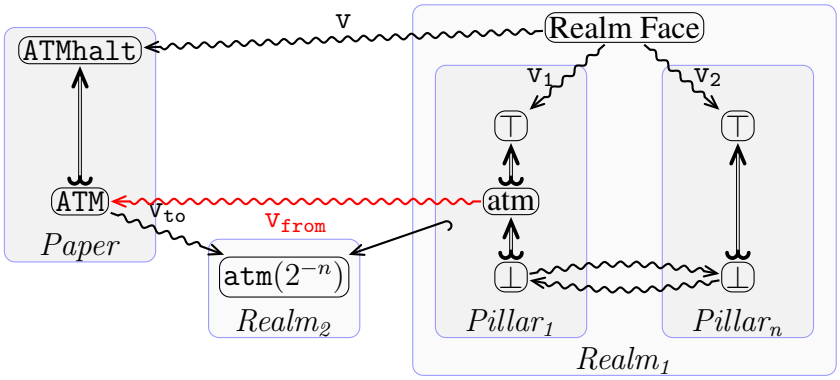


Figure 6.17: Publication Graph for Specialization Recaps (Example 2.41)

Multiple Recaps Up to now we have only treated cases with single recaps to ease the exposition. But papers and especially textbooks often recap from different realms and base the rest of the exposition on them.

This is the situation in Figure 6.18a inspired by Examples 2.42 and 2.43 from Rudin's book. Note that the two recaps import directly from the faces rather than from a specific pillar or paper. This is intended and covers the typical case of recaps in textbooks and survey articles. For instance, as mentioned in Section 2.1 (**P8a**), Rudin does not directly cite literature in the recaps, but aggregates the vast literature in an appendix.

For the aggregation phase the multiple recaps situation begs the question of where the contribution should be placed. In the common ground part of Rudin's book we have separate recaps of vector spaces and topological spaces (2.42), and we analyze them as theory morphisms from their respective realms. In this case, there is the realm of topological vector spaces (2.42) which imports from both realms, this is the natural place for the contributions. In the case such a realm does not exist yet, the paper can be used as the natural starting point for (first pillar of) the realm. Actually, the "union realm" concept in Figure 6.18b is a bit simplified. The contribution of the paper will usually add some conditions – like conditions (a) and (b) in (2.43) – and use that for

the base theories of the realms. This does not invalidate our claim that there is always a natural realm – which may have to be created – for the contribution of the "paper".

6.1.6 Foundational Ambiguity

As observed in Section 2.1 (**P8b**), establishing common ground in papers has two parts. We discussed one of them, recaps, above in Section 6.1.5. However, the other, foundational ambiguity, while similar in most respects, is more complex. Foundations are not typically explicitly mentioned, and papers routinely use concepts and theorems without analyzing the foundational assumptions they require. It is then left to the reader to infer an appropriate foundation in which the results from the paper work. To an extent, this is similar to recaps above, as papers built upon a aggregation of foundational assumptions from various logics or set theories. However, in the case of foundations, the premise of equivalence between them does not hold. Some foundations are more expressive than others and that expressivity may or may not be used in a particular paper.

Therefore, to represent foundational ambiguity we generalize the concept of realms introduced above to the case where the pillars are not necessarily isomorphic. Concretely, we allow having (total) views in only one direction, in which one pillar is a special case of another. Then, the face has the expressivity of the most general pillar which gives a powerful set of concepts and axioms for library developers. However, in case the full expressivity is not used, some results might hold in the weaker pillars also and there are heuristics to determine that. Therefore, the foundation can be established ambiguously as the face of such a pre-realm, and, possibly, refined later to a concrete pillar.

Pre-realms The intuition is that pre-realms are realms where for each pair of views v_l, v_r between two pillars, one of them may be partial. Partial views are described in [RK13a] and we refer to that for more details. However, the central idea is that the requirement that there must exist an assignment for *every* undefined constant in the source theory is dropped. This implies that the empty view containing no constant declarations is always a valid partial view.

Concretely a pre-realm is a structural feature $\langle \mathfrak{PreRealm}, IF, 0, \mathcal{V}, \mathcal{E} \rangle$. Its pragmatics are either instances of the $\mathfrak{Pillar}$ or $\mathfrak{PartialMorph}$ features. $\mathfrak{PartialMorph}$ is a binary subfeature where its two arguments are references to the two views (one of which may be partial). It corresponds to $\mathfrak{Isomorphism}$ subfeature for realms. The validity predicate $\mathcal{V}$ ensures that these well-formedness conditions hold.

The face generation algorithm remains unchanged, but the consistency of the resulting theory is no longer guaranteed by IMMT. In fact, the resulting theory may often be inconsistent. This conforms, however, with mathematical practice where the foundation is typically unspecified. Instead, the author assumes as a foundation any and all theories where the necessary concepts and theorems exist or can be derived. As mentioned in Section 2 the situation is particularly interesting since the individual foundational options may be incompatible with each other.

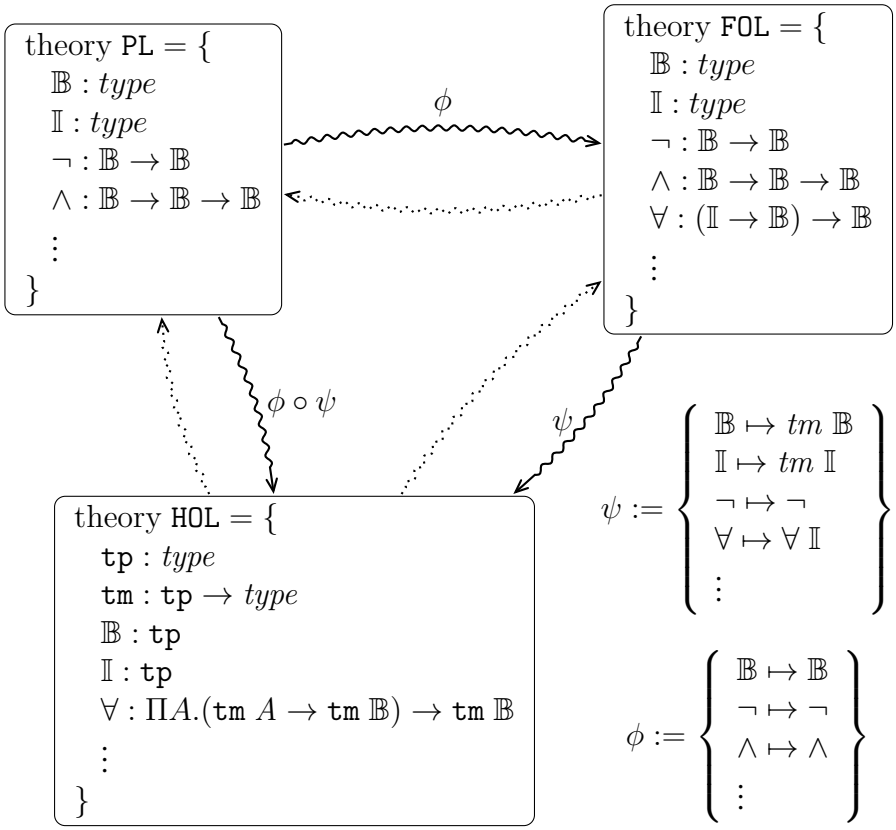


Figure 6.19: PL, FOL and HOL in IMMT

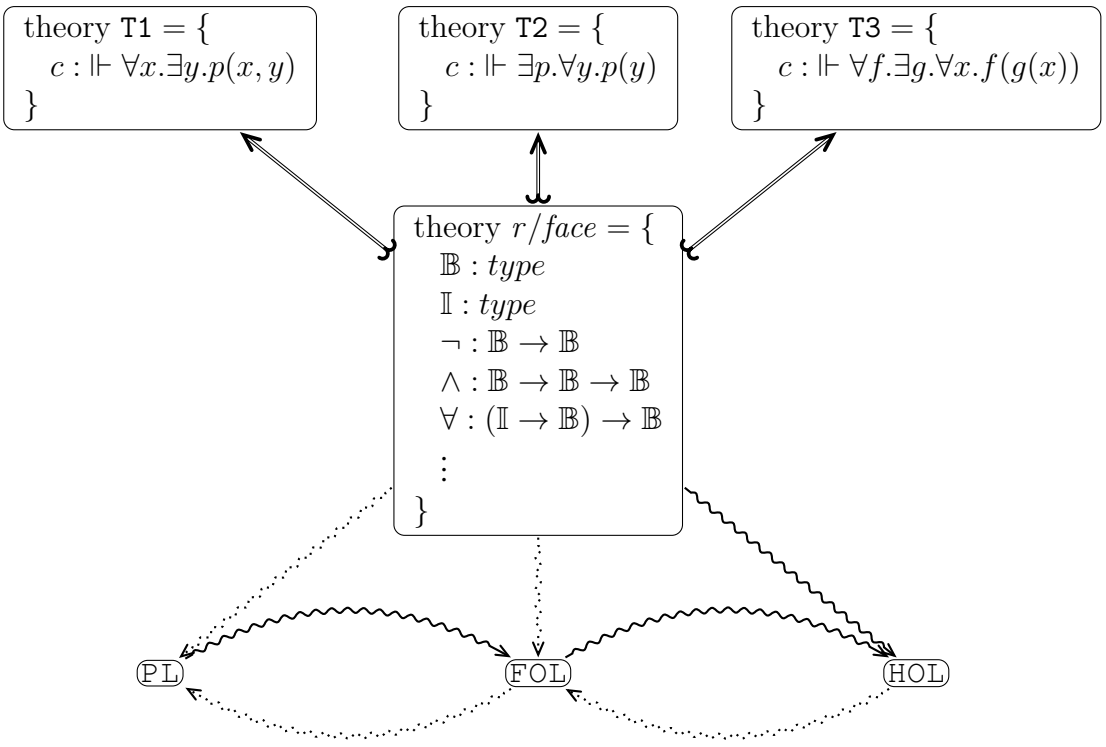


Figure 6.20: A pre-realm in IMMT

Example 24. Figure 6.19 shows a IMMT development of propositional logic, first-order logic and higher-order logic. We omit much of the details for brevity but refer to the formalizations from the LATIN project [CHK+11] for details. The views ϕ from PL to FOL and ψ from FOL to HOL are total and straightforward. The reverse ones ϕ^{-1} from HOL to FOL and ψ^{-1} from FOL to PL are also straightforward but partial. Therefore the three logics do not form a realm but they do form a pre-realm.

Figure 6.20 shows the pre-realm formed from the three logics together with three developments on top of it. The first one only uses PL-induced concepts and theorems and therefore is valid in all three logics. The latter two use quantifiers and higher-order quantifiers and are therefore only valid in FOL or HOL and, respectively, just HOL. By inspecting the theorems themselves we can deduce the valid logics, but the development happens in a possibly inconsistent union-like theory that offers maximum expressivity. This conforms with the way development happens in actual mathematical practice where the necessary foundational assumptions are not made explicit at the beginning of the development process, if at all.

Pre-realms are useful because they allow library developers using IMMT to do mathematics in a manner similar to current mathematical practice. They can work in an environment where a rich tool-chest of primitive and derived concepts as well as axioms and theorems is available to them. Then, after they *develop* concrete results, they can *refine* their developments to use a consistent foundation with minimal assumptions. The refinement process can benefit from

algorithmic support such as dependency analysis (see [IR12a] for more details) but we leave that as future work. However, we emphasize that, currently, developing good theory graphs or refactoring existing ones is a very difficult, tedious process. This is a major obstacle in producing large libraries of deeply interconnected, theory-graph structured formal libraries. We believe that pre-realms can be a major contributor to simplifying this not only by supporting a more natural development process. But, also, because they allow separating development from the refinement process. Then, for instance, while developers produce new results, we can have dedicated librarians that oversee the refinement process in organizing the library. However, at this point, we leave that discussion for future work.

6.2 Formal Mathematics

Acknowledgement. *Part of the results in this Section, in particular in Sections 6.2.1, 6.2.2, and 6.2.3 are based on collaborative work with Michael Kohlhase and Florian Rabe and have been prepared for publishing in [IKR].*

Narrative information and a rich set of structuring constructs are not exclusive to informal mathematics. In fact, as discussed in Section 2.2, narration and structure are common in formal mathematical languages and libraries. Most formal languages have their own primitives for representing statement-level constructs such as definitions, theorems or lemmas as well as module-level elements such as documents or sections. While formal content is checked by computers, it is often written and read by humans which benefit from organizing knowledge using such constructs. Therefore, while typically machine-processed, formal mathematics is (partially) narrative in practice.

As already discussed, the main function of structural features is to allow language designers (logics, logical frameworks) in IMMT to declare and implement their own declaration-level constructs. This means that structural features and IMMT also allow for a more adequate representation of formal languages and their libraries.

In this Section we define a set of basic such constructs that are common in other formal languages. In the process, we develop an initial kernel of structural features that language designers can reuse and extend. We also recover the relevant parts of MMT's original expressiveness by re-defining MMT-style documents and structures in IMMT using structural features.

6.2.1 Local Scopes

Structuring mathematical developments by grouping similar or related declarations together is common in both informal and formal mathematics. For example, LaTeX uses sectioning and environments, OMDOC uses `<tgroup>` elements, and many proof assistants use modules.

The structural feature $\mathfrak{Scope}$ of arity 0 yields the simplest special case. It is defined by $\Theta \vdash_T s =$

$\mathfrak{Scope}\ \{\Sigma\} \rightsquigarrow [s/\_]\Sigma$. Here the strict declarations are the same as the pragmatic ones except for qualifying them with the name of the scope.

6.2.2 Sections

Sections are similar to local scopes except they may contain local variables, i.e., constants which are in scope only inside the section but not visible from the outside. This is well-known from informal mathematics and also part of some proof assistants like Coq (which allows declaring `Variables` in a `Section`). The idea is that some constants (the local variables) (i) behave like normal constants from the perspective of the pragmatic declarations, (ii) but do not appear among the strict declarations because all of them abstract over local variables.

We use a structural feature $\mathfrak{Section}$ with arity 0 and define $\Theta \vdash_T s = \mathfrak{Section}\ \{\Sigma_0\} \rightsquigarrow \Sigma_2$ as follows. First, let Σ_1 be the recursive elaboration of Σ_0. We define Σ_2 by induction on the declarations in Σ_1:

- For every constant without definiens, nothing. These are the local variables.
- For every constant declaration C of the form $c[\ :\ E_1] = E_2$, let Γ be the context containing all local variable declarations that precede C. Then Σ_2 contains $c[\ :\ \Pi\ \Gamma.\bar{t}(E_1)] = \lambda\ \Gamma.\bar{t}(E_2)$ where t is defined below.

Abstracting away local variables is not foundation-independent: It requires a foundation with a Π and λ binder such as LF. This is necessary to bind the local variables as seen above. Thus, the feature $\mathfrak{Section}$ must be defined in a foundation that imports a theory for a dependently-typed λ-calculus.

Moreover, because every constant c elaborates to a function that takes the local variables as arguments, every reference to it must be applied to these local variables. This is the purpose of the translation t: It translates every reference to a Σ_1-constant c that is not a local variable to $@\ c\ c_1\ \ldots\ c_n$, where the c_i are the names of the constants in Γ. Here $@$ is the constant for function application that goes with Π and λ.

There are multiple variants of this feature. Firstly, Σ_2 does not use the name of the section as a qualifier for the strict declarations. Alternatively, we could qualify the declarations as with $\mathfrak{Scope}$. Secondly, Σ_2 abstracts over all local variables that precede a constant declaration C. Alternatively, we could abstract only over those that actually occur in C. Thirdly, we could also use local definitions, i.e., specially marked defined constants that are eliminated during elaboration, namely by expanding their definitions.

6.2.3 MMT Structures

Structures are a primitive of the MMT language [RK13a], where their name is inspired from the SML module system. We can now recover them as a special unary structural feature $\mathfrak{Structure}$.

The intuition behind the elaboration function $\Theta \vdash_T s = \mathfrak{Structure}\ E\ \{\Sigma_1\} \rightsquigarrow \Sigma_2$ is that

- E is an expression evaluating to some theory, say with body Σ_a,
- Σ_1 is then interpreted as a partial morphism from Σ_a to T, say with domain Σ_b,
- adding Σ_2 to T yields the pushout of Σ_1 and the inclusion $\Sigma_b \hookrightarrow \Sigma_a$.

Concretely, Σ_2 arises from $[s/\_]\Sigma_a$ by merging in all declarations in Σ_1. For example, if Σ_a contains $c : E_1$ and Σ_1 contains $c = E_2$, then Σ_2 contains $s/c : [s/\_]E_1 = E_2$. We refer to [RK13a] for the details.

6.2.4 Documents

In MMT documents are primitives constructs and are effectively defined as a named list of modules or module references. In IMMT we omit the document layer (see Section 5.2) because it can be recovered in greater expressivity using structural features. Concretely, we use derived modules to represent documents because they elaborate to lists of modules. Therefore, we can define dedicated features for each particular kind of document and then define associated subfeatures to represent the document elements.

For example we can define OMDOC-style documents [Koh10] using a structural feature $\mathfrak{ODoc}$ together with the following subfeatures:

- a nullary structural feature $\mathfrak{OGroup}$ corresponding to the OMDOC `omgroup` element which is used for sectioning. Its pragmatics are either strict modules or other instance of $\mathfrak{OGroup}$. Then, the elaboration function returns the strict modules as given and elaborates nested instances of $\mathfrak{OGroup}$ recursively.
- three unary structural features $\mathfrak{OToc}$, $\mathfrak{OBiblio}$ and $\mathfrak{OIndex}$ corresponding to the OMDOC `tableofcontents`, `bibliography` and, respectively, `index` elements. They each have empty pragmatics and elaboration as they only needed by IMMT exporters (i.e. presenters) when preparing the document for presentation (e.g. as HTML to be shown in a browser). The single argument encodes parameters for the respective elements as defined in [Koh10] and may be used by exporters to adjust presentation (e.g. bibliography style).

The validity predicate checks that the pragmatic declarations are either IMMT strict modules (i.e. theories or views) or instances of the subfeatures given above. The elaboration is recursive, meaning that it is defined as the concatenation of the elaborations of each document element.

OMDOC documents are just one example of documents that can be represented using IMMT structural feature and derived modules. In Chapter 7 we discuss concrete importing existing mathematical libraries into IMMT and, in the process, define IMMT structural features for representing Mizar and, respectively, OEIS (Open Encyclopedia of Integer Sequences) documents.

Note that a key feature of the IMMT language is that we can use narrative URIs (using the "!" separator) to refer to the individual document elements (i.e. pragmatics in IMMT). This is important on its own as it allows referring to previous narrative elements within the document which is particularly relevant for large documents such as books or course notes. The need

for such a referencing mechanism is discussed more in [Koh15] which served as one of the motivations and design goals for IMMT URIs.

Part III

Flexiformal Knowledge Management

Chapter 7

Flexiformal Mathematical Libraries

In this Chapter we describe a case study in building a large flexiformal library of mathematical knowledge. We realize this by importing several pre-existing mathematical libraries into IMMT. To obtain a comprehensive library and to better evaluate the IMMT language we choose libraries at different levels of formality. Concretely, we import 1. the Mizar Mathematical Library (MML) [TB85, MML] as a fully formal corpus, 2. the STEX-based SMGloM and GenCS libraries [Koh16, GIJ+15] for the semi-formal case, 3. and the Open Encyclopedia of Integer Sequences (OEIS) [OEI15] for the informal case.

For each of the above-mentioned libraries, the process of importing them into IMMT is similar and involves two steps. First, we design an IMMT *foundation* that declares the primitive symbols (as MMT constants) and language constructs (as IMMT structural features) of the target language. Second, we implement an IMMT *importer* that processes files from the original corpus (or an export of it) and produces IMMT content. It uses the foundation above whenever primitive symbols or language constructs appear.

Figure 7.1 shows an overview of the resulting library as an IMMT theory graph. Note that IMMT and the logical frameworks are technically two separate levels as frameworks are declared as theories in IMMT. However, we conflate them in 7.1 for simplicity. We declare $Mizar$, $sTeX$ and ISF (Integer Sequences Foundation) as IMMT theories corresponding to the base languages of the libraries discussed above. The informal foundation IF developed in the previous Chapter opens the door for importing a more conventional informal library such as ARXIV. However, this remains future work at this point. Once the foundations exist, we import their libraries to integrate the mathematical knowledge represented in them into one (IMMT) system and language. Since the libraries are at different levels of formality we effectively create a large flexiformal library of mathematical knowledge represented in IMMT. This permits interoperability between the libraries and also enables better machine support. Additionally, the key novelty of the IMMT imports is that we can now adequately represent the primitive language constructs using structural features. That means we can provide more accurate, structure-preserving imports, instead of de-constructing the knowledge by discarding narrative and structural information. Such information is essential for building valuable, user-facing services and applications for (or based on)

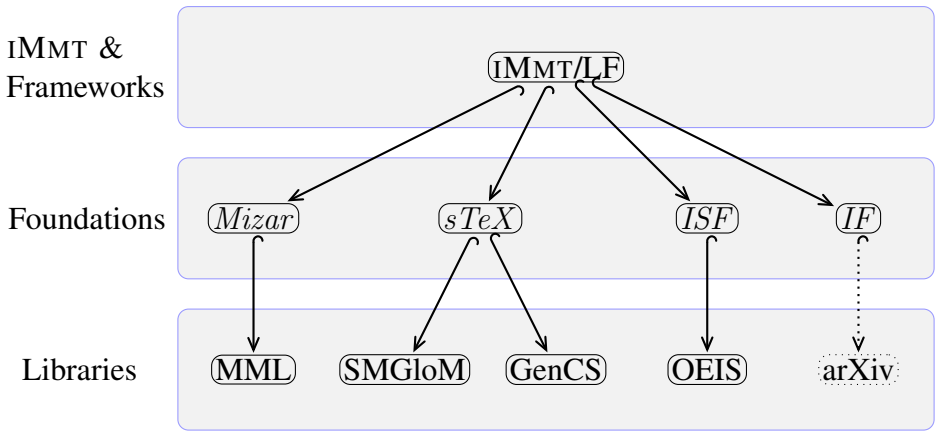


Figure 7.1: Overview of the IMMT Flexiformal Theory Graph

these libraries. We discuss the applications perspective in more detail in Chapter 8.

Below we discuss each of the aforementioned languages and imports individually. We focus on the narrative, structural aspects as this represents the key representational novelty enabled by the IMMT system and language.

7.1 Importing the Mizar Library into IMMT

Acknowledgement. *The results in this section are based on collaborative work with Michael Kohlhase and Florian Rabe as well as Joseph Urban that was published in [IKR11] and, respectively [IKRU13]. However, here, we focus on the work done by the author and significantly revise and extend it to re-frame it in terms of* IMMT *and structural features.*

As introduced in Chapter 3 and Section 2.2, Mizar is a formal language for representing mathematics whose design goal is to stay as close as possible to the mathematical vernacular. A consequence of this goal that is relevant here is that the Mizar language has a breadth of interesting statement-level constructs to capture mathematical intuitions. The Mizar community develops and maintains a library of mathematical knowledge in the Mizar language called the Mizar Mathematical Library (MML). Currently, The MML consists of over 1000 articles that contain over 50000 theorems and around 10000 definitions.

In this Section, we present an import of the Mizar Mathematical Library into IMMT and focus on representing the rich set of statement-level constructs used to represent mathematics in Mizar and the MML. For a full overview of the statement-level constructs of Mizar and their representation in this import see [IKR11] and for a more concise and high-level description we refer to [IKRU13]. The base logic of Mizar which we use to ground our import is in the LATIN library and is described in [IR11].

To represent Mizar articles we define a dedicated structural feature $\mathfrak{MizArticle}$ with a series of

subfeatures corresponding to Mizar statement-level constructs. Then the validity function for $\mathfrak{MizArticle}$ checks that its pragmatic declarations are all instances of its subfeatures. Finally, the elaboration function, simply elaborates each pragmatic declaration recursively. We give the central subfeatures below and give their elaboration inline as part of their definition.

Justified Theorem Justified theorems are propositions together with a proof of correctness. For example, the listing below, shows a proof that if a set is not the empty-set ($\{\}$ in Mizar) then it has at least one element.

```
1 theorem
2       X <> {} implies ex x st x in X
3       proof
```

We represent this by defining a binary structural feature $\mathfrak{JustThm}$ where the first argument E_1 is the proposition and the second argument E_2 is the proof. The validity constraints require that the proof given has the correct type, namely that E_2 is of type $\Vdash E_1$.

Then, a derived declaration $t = \mathfrak{JustThm}\ E_1\ E_2\ \{\cdot\}$ elaborates to a statement $t : \Vdash E_1 = E_2$.

Definition Definitions in Mizar can be used to introduce *predicates*, *functions*, *attributes*, *modes* and *structures*. The first four are structurally very similar and the distinctions between are mostly at the level of the underlying logic (which is discussed in [IR11]). For instance, they require specific primitives such as the description operator for implicit definitions or predicate types for attributes and modes. However, Mizar structure definitions are more complex because they can contain several internal declarations (i.e. pragmatics). This distinction corresponds to the plain statements versus structured statements observation from Section 2.1.

Simple Definitions give the type of the concept being defined and its definition. The listing below shows an example predicate definition that introduces the subset relation (`c=` in Mizar). The `let` keyword binds the local variables for the definition and also determines the type. Since there are two arguments, and `c=` is a predicate, its type is `set → set → bool`.

```
1 definition
2       let X, Y be set ;
3       pred X c= Y means
4       for x being set st x in X holds x in Y;
5 end;
```

We define a binary structural feature $\mathfrak{MizDef}$ where the first argument E_1 is the type and the second E_2 is the definition. The validity constraints require a well typing relation between E_1 and E_2. Then, a derived declaration $d = \mathfrak{MizDef}\ E_1\ E_2\ \{\cdot\}$ elaborates to a statement $d : E_1 = E_2$.

Structure Definitions follow the same pattern as the others but are structurally more complex as, internally, a structure definition has declarations for each selector. One can think of structures as Mizar's implementation of record types or, from an informal perspective, mathematical structures. From that perspective, the selectors represent the members of the structure (fields of

a record type). Structure definitions may define new selectors as well as extend existing structures. For example, the structure definition below defines a new structure `2-sorted` that extends `1-sorted` and adds a new selector of type `set` namely `carrier'`.

```
definition
struct (1-sorted) 2-sorted(#carrier,carrier' -> set#);
end;
```

Therefore, we define first two unary subfeatures $\mathfrak{StructRef}$ for extending an existing structure and $\mathfrak{Selector}$ for declaring a new selector. For the former, the argument is a reference to the structure being extended. For the latter, the argument is the type of the selector.

Then, we can define $\mathfrak{MizStruct}$ as a nullary structural feature. Its validity function requires that its pragmatics are either instances of $\mathfrak{StructRef}$ or or $\mathfrak{Selector}$. Then a derived declaration $s = \mathfrak{MizStruct}\ \{\Sigma\}$ elaborates to a set of declarations Σ' be elaborating each declaration d in Σ as follows:

- if d is a $c = \mathfrak{Selector}\ E\ \{\cdot\}$ we add $s/c : E$ to Σ'
- if d is a $c = \mathfrak{StructRef}\ s_2\ \{\cdot\}$ we add $s/c_i : E$ to Σ' for each selector $c_i : E$ of the structure s_2

Notation In Mizar, notations introduce a new name for a pre-existing symbol and may be synonymic or antonymic. For example, as seen in the listing below, one can denote *odd* as an antonymic notation of *even* (provided *even* is previously defined):

```
notation
      let i be Integer;
      antonym i is odd for i is even;
end;
```

We define two unary structural features $\mathfrak{MizSyn}$ and $\mathfrak{MizAnt}$ where the argument is the reference to a symbol and the validity constraints require the reference to be valid. Then, a derived declaration $n = \mathfrak{MizSyn}\ s\ \{\cdot\}$ elaborates to a constant $n = s$. Correspondingly, a derived declaration $n = \mathfrak{MizSyn}\ s\ \{\cdot\}$ elaborates to a constant $n = \neg s$

Note that the pragmatic declarations that encode the Mizar constructs and their structure roughly corresponds to the Mizar pattern level. Meanwhile, the strict (elaborated) declarations roughly correspond to the constructor-level syntax. The former is useful for users and applications that can understand and make use of structure. The latter is simpler and is useful for applications such as type and proof checking or for meta-theoretical reasoning. This shows how IMMT can be used to capture dual aspects of mathematical knowledge not just in informal but also in formal mathematics. While we omit some details here for brevity, we were able to represent all significant statement-level language constructs of Mizar using structural features.

Implementation The Mizar importer is implemented in Scala as an IMMT extension and is available at `https://github.com/KWARC/MMT/tree/master/src/mmt-mizar`. It

contains about 4100 lines of code and over 200 classes. First, Mizar runs over the whole library of `miz` files and produces the Mizar `xml` files which serve as input for our importer. Our code parses the `xml` files and then uses the resulting data to construct the corresponding IMMT declarations (e.g. theories, constants, bipartite or derived declarations). This takes about 2 hours on an Intel $i5$ processor and 16GB of RAM and results in over 1GB of data. The parsing algorithm is tedious but straightforward. We omit the details here and refer to [IKR11] and the importer code itself for details.

7.2 Importing STEX Libraries into IMMT

STEX [Koh08, sTe] is a LATEX-based surface syntax for OMDOC that is optimized for narrative-oriented, semi-formal mathematical content. The language comes with a parser implemented as a plugin [SLP] for the LATEXML system [Mil] which produces STEX-flavored OMDOC files.

In the MathHub system [IJKW14], we have over 5000 semi-formal STEX source files split over several libraries. Most notably, they contain 3000 files from course notes introducing basic concepts of computer science and another 2000 files from the SMGloM project (Semantic Multilingual Glossary of Mathematics) [GIJ+15, SMG] containing definitions in multiple languages for important mathematical concepts.

We use the STEX content as a representative example to evaluate our language with respect to representing semi-formal content. Additionally, due to STEX being a surface syntax for OMDOC, we also evaluate IMMT with respect to the original goal of recovering OMDOC's expressivity on top of the MMT language.

7.2.1 Object Level

STEX declarations are formed using `CMP` and, occasionally, `FMP` elements. `CMP`'s or *commented mathematical properties* are effectively a native format (typically HTML) mixed with OPENMATH terms. `FMP` elements or *formal mathematical properties* are OPENMATH terms. In terms of IMMT objects, `CMP`'s and `FMP`'s naturally correspond to the narrative and, respectively, to the content aspect. In fact, since the terms are represented using OPENMATH which is subsumed by IMMT terms, their representation in IMMT is straightforward.

In the common case where the `FMP` is missing, we extract the objects from the `CMP` to obtain a more interesting content term. Them in the IMMT import, each term from original the `CMP` is added as an argument to the opaque term, and replaced with a matching realization spec in the narration (see Example 25 below).

Example 25. Consider a definition d with no `FMP` and the `CMP` being:

$$L_0\ E_1\ L_1\ E_2\ \ldots\ E_n\ L_n$$

where $\vec{E}$ are terms and $\vec{L}$ are plain text literals (i.e. strings). Then, the corresponding IMMT term is :

$$L_1\ \rho(1;\cdot)\ L_2\ \rho(2;\cdot)\ \ldots\ \rho(n-1;\cdot)\ L_n \parallel \bullet[\cdot](\vec{E})$$

7.2.2 STEX primitives as IMMT declarations

Symbol Declarations In STEX, symbol declarations introduce a new symbol with a name but without a type or definiens. Definitions (discussed below) are then standalone declarations that can *define* one or more pre-declared symbols. Therefore, we represent STEX symbol declarations as plain IMMT constants with no type or definiens.

For instance, the STEX symbol declaration `\symi{subset}` (OMDOC shown in listing below) corresponds to a plain IMMT constant $subset$.

```
1 <omdoc:symbol name="subset" stex:srcref="smglom/sets/source/subsupset.tex#
     textrange(from=4;0,to=4;48)"/>
```

Notations STEX notations, or more generally OMDOC notations are defined by two child elements, `prototype` and `rendering`. The prototype is an OPENMATH object with dedicated elements for the arguments. The rendering is MathML with dedicated elements for referring to the arguments from the prototype. From the perspective of IMMT notations presented in Chapter 6 the prototype corresponds to the content representation and the rendering to the narration. The argument placeholders in the rendering (`omdoc:expr`) are represented as IMMT variables with their name attribute used as the variable name. Then, the references in the rendering (`omdoc:render`) are represented as realization specs with their additional attributes (e.g. precedence) added to the presentation context. The main difference, is that `omdoc:render` elements use the name attribute for referencing, in IMMT we translate that to a position instead. Then, the translation to IMMT becomes straightforward. This is expected since IMMT notations were designed based on OMDOC notations. Example 26 below shows the IMMT representation of a simple OMDOC notation from the SMGloM library.

Example 26. The listing below shows the OMDOC notation $x \subseteq y$ for the symbol *subset* from the theory *subsupset*[1]:

```
1 <omdoc:notation cd="subsupset" name="subset" stex:srcref="smglom/sets/source
     /subsupset.tex#textrange(from=5;0,to=5;63)">
2   <omdoc:prototype>
3     <om:OMA>
4       <om:OMS cd="subsupset" cr="fun" name="subset"/>
5       <omdoc:expr name="arg1"/>
6       <omdoc:expr name="arg2"/>
7     </om:OMA>
8   </omdoc:prototype>
```

[1]The source file used in this example is available at `https://gl.mathhub.info/smglom/sets/blob/master/source/subsupset.tex`.

```
9   <omdoc:rendering precedence="300">
10    <omdoc:render name="arg1" precedence="300"/>
11    <m:mo cr="fun">⊆</m:mo>
12    <omdoc:render name="arg2" precedence="300"/>
13  </omdoc:rendering>
14 </omdoc:notation>
```

The prototype on lines 2-8 contains the applied symbol with `omdoc:expr` elements for the placeholder arguments. The `omdoc:expr` are given (generated) unique names, `arg1` and `arg2` in this case. Then, the names are used in the rendering to refer to the arguments. The rendering (lines 9-13) is MathML with placeholders for the argument references. They are represented using `omdoc:render` elements on lines 10 and 12 both of which additionally set the argument precedence to 300. The corresponding IMMT notation term is:

$$P^O \parallel (subsupset?subset\ arg1\ arg2)$$

where the presentation P^O is:

$$\rho(1;\langle prec \mapsto 300\rangle)\ \texttt{<m:mo cr="fun">⊆</m:mo>}\ \rho(2;\langle prec \mapsto 300\rangle)$$

Definitions STEX definitions are standalone, declaration-level primitives that introduce a definition for one or more pre-declared symbols. Separating symbol declaration from definitions has several representational advantages. First, symbols can have more than one definition. Second, definitions can be added later, even in another theory, refining the meaning of pre-existing symbols. Finally, the same definition can define more than one symbol, which happens often in narrative math. For instance, defining concepts together with their components such as graph together with edge and vertex. Similarly dual concepts such as prime and composite numbers are often introduced together.

In IMMT we represent STEX definitions as derived statements while the OMDOC symbols being defined are IMMT statements and typically constants. Concretely, we define a flexary structural feature $\mathfrak{Definition}$ where its term parameters represent references to the statements being defined followed by a term encoding the definition as a proposition about them. The validity predicate ensures the references are valid and that the list of pragmatics is empty.

Then, the elaboration is defined as $\Theta \vdash_T d = \mathfrak{Definition}\ \vec{E}\ E_1\ \{\cdot\} \rightsquigarrow \{\Sigma\}$ where Σ contains the single statement $d : \Vdash E_1$. Therefore, semantically, the definition is seen as an axiom involving the defined symbols. However, the definitional nature encoded by the feature is used by other IMMT services such as the QMT [Rab12] query language (which uses an index of relations). Therefore we can add the appropriate `isDefinedBy` binary relations for each instance of $\mathfrak{Definition}$. Then, IMMT can correctly answer QMT queries about definitions, which are used in several services, most importantly in the definition lookup service. We discuss this in more detail in Chapter 8.

Example 27. The listing below gives an STEX definition for the symbol `subset`. The `for` attribute gives the target being defined and the `omdoc:CMP` child gives the body of the definition.

```
<omdoc:definition for="subsupset?subset" xml:id="subset.def" >
  <omdoc:CMP>
    <p>A set <om:OMOBJ><om:OMV name="A"/></om:OMOBJ> is a
      <omdoc:definiendum cd="subsupset" name="subset">subset</omdoc:
         definiendum> of a set
      <om:OMOBJ><om:OMV name="B"/></om:OMOBJ> (written
        <om:OMOBJ><om:OMA><om:OMS cd="subsupset" name="subset"/><om:OMV name
           ="A"/><om:OMV name="B"/></om:OMA></om:OMOBJ>),
      iff all <om:OMOBJ ><om:OMA><om:OMS cd="set" name="inset"/><om:OMV name
         ="x"/><om:OMV name="A"/></om:OMA></om:OMOBJ>
      are members of <om:OMOBJ><om:OMV name="B"/></om:OMOBJ>.
    </p>
  </omdoc:CMP>
</omdoc:definition>
```

For the IMMT representation we use the `xml:id` attribute as the IMMT name of the declaration. Then, the IMMT constant is : $subset.def$: $\Vdash E$ where E is the IMMT representation of the `omdoc:CMP` element above.

Other STEX declarations such as *Examples* or *Exercises* are treated analogously with two additional subfeatures $\mathfrak{Example}$ and $\mathfrak{Exercise}$. They are semantically represented as constants while their narrative structure is used by the query service and, therefore, dependent applications.

Proofs To represent STEX proofs, we reuse the $\mathfrak{Proof}$ structural feature introduced in Section 6.1.3. Since it covers OMDOC proofs and STEX is a surface syntax for OMDOC it proves sufficient to represent the proofs encountered in STEX libraries.

Glossary Modules Glossary modules are not primitive STEX constructs but are a central part of the data model for the SMGloM library described in [Koh14a]. There, narrative definitions in different languages for glossary entries are organized into *glossary modules* with the implicit contract that the definitions are semantically equivalent. Here we frame the data model described in [Koh14a] as a special case of a realm.

Glossary modules consist of a *signature* theory that declares the concept being defined and a set of *language bindings* that declare its definition. There is exactly one binding for each language. The implicit requirement is that the definitions in each language are semantically equivalent. Therefore, the entire module can be seen semantically as one concept with alternative definitions.

Figure 7.2 shows a glossary module M as a realm. The signature theory $M.sig$ is a common base theory that lives outside the realm. Then, each language binding $M.L_k$ is a pillar consisting of exactly one theory. Finally, the face corresponds to the intuitive view of the glossary module as one concept with alternative definitions.

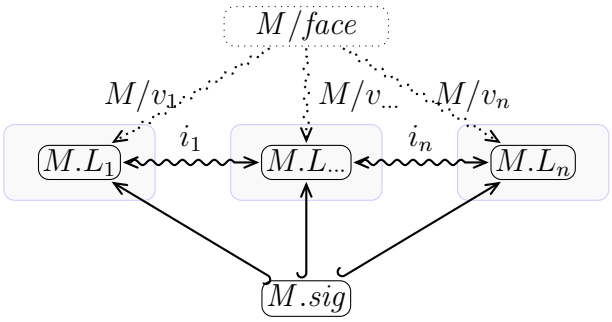


Figure 7.2: SMGloM modules as Realms

7.2.3 Implementation

Like in the Mizar case, the sTEX importer is implemented as an iMMT extension and does not directly parse the source files but an XML syntax produced after some pre-processing. In the sTEX case, the pre-processor is LaTeXML which returns the standard OMDoc-XML syntax. The parsing algorithm is mostly straightforward following the OMDoc specification and then the iMMT data structures are constructed as described above. The source code for the importer is available at `https://github.com/KWARC/MMT/tree/master/src/mmt-stex` and contains about 2400 lines of code and around 40 classes.

It parses the files while checking well-formedness with respect to iMMT-level validity. Both the sTEX content and format itself are currently under active development and there is no other full checking of the well-formedness of the content. Therefore, importer errors can be caused by either source errors, issues with the LaTeXML sTEX-plugin or finally issues with the sTEX-importer itself. As a result, evaluating the correctness of the importer can be difficult because the resulting errors often have to be manually sorted into the three categories by inspecting the relevant files. In fact, during and after its development the importer became the de-facto checker for sTEX source files and was used to improve all components of the compilation workflow (see Section 8.2 for more details).

7.3 Importing the OEIS Library into iMMT

Acknowledgement. *The results in this section are based on collaborative work with Enxhell Luzhnica and Michael Kohlhase that was published in [LIK15]. However, we only present here the part of the work done by the author and refer to [LIK15] for the rest.*

The On-line Encyclopedia of Integer Sequences (OEIS) [OEI15] is a publicly accessible database storing and documenting sequences of integers that are mathematically or practically meaningful. The effort was started in 1964 by N. J. A. Sloane and led to a book [Nei73] describing 2372 sequences which was later extended to over 5000 in [NS95]. The online version started in 1994 [Nei94] and currently contains over 250000 documents from thousands of contributors

with 15000 new entries being added each year [Nei12]. Documents contain various information about each sequence such as the beginning of the sequence, its name or description, formulas describing it, or computer programs in various languages for generating it.

The OEIS database is stored as a collection of text documents (one for each sequence) written in the *internal format* of OEIS. The internal format only defines the document-level structure of the sources. Specifically, it provides dedicated markup to distinguish between text snippets that represent different kinds of information about the sequence (e.g. formulas, program code, examples or references). Therefore, parsing the *document structure* is straightforward. Inside the actual text snippets the format is not standardized which makes parsing them, particularly the formulas, non-trivial. Still, in practice, the syntax used in the text snippets is somewhat regular and, in [LIK15], we were able to build a *formula parser* that succeeds on most formula snippets in the OEIS.

In the context of this thesis the OEIS library is interesting because it is natively informal, but can be partially semanticized (i.e. by the formula parser). Therefore, it provides a very rough blueprint for a (fragment of) a more typical library of informal mathematics such as ARXIV. In that case, more complex semantic analysis and natural language processing tools would for semanticizing but the general work-flow would be similar.

The OEIS document format The *internal format* [OEI] is line-based in the sense that each line starts with a marker that represents the kind of content found in that line. We briefly introduce the relevant kinds below but refer to [OEI] for details.

1. The *identification* line gives the unique ID of the sequence declared in that document.
2. The *start values* lines give the beginning of the sequence. It is used for searching as well as lexicographic ordering of the sequences.
3. The *name* line gives the name, a brief description or the definition of the sequence.
4. The *references* lines contain references to papers which in turn refer to or are concerned with the sequence described in the current document.
5. The *formula* lines give formulas that define or hold for the sequence described in the current document. The formulas are in plain text ASCII syntax that is similar to LaTeX math markup and can contain text as part of the formula or as comments.
6. The *author* line gives the document author typically containing the name and e-mail address.
7. The *examples* lines give examples and additional information about the initial terms of the sequences

The actual document-level syntax of the internal format is presented in Figure 7.3 where we use $\underline{\circ}$ for required lines, $[\circ]$ for optional parts, $\circ^*$ for repetitions and $\circ_1 \mid \circ_2$ for alternatives. Moreover we use *italic* for grammar productions, `typewriter font` for fixed syntax and normal font for informal descriptions.

The import is relatively straightforward once the formulas have been parsed. First we define a OeisDoc structural feature for representing OEIS documents, then we define subfeatures for its

Ident.	::=	%I *ID* $[M_n]$ $[N_n]$
ID	::=	Identification number in [OEI15]
M_n	::=	Identification number in [NS95]
N_n	::=	Identification number in [Nei73]
Values	::=	%S *ID* nr* [%T ID nr* [%U *ID* nr*]]
Name	::=	%N *ID* text
References	::=	%D *ID* text
Formula	::=	%F *ID* formula
Author	::=	%A *ID* text
Examples	::=	%E *ID* text

Figure 7.3: Grammar of OEIS Internal Format

pragmatics in correspondence with the kinds of lines described above. For each OEIS document, we use the OEIS sequence ID as the name of the derived module instantiating the $\mathfrak{OeisDoc}$ feature. Then the subfeatures are straightforward following the definitions of the lines given above :

1. The identification line is a subfeature $\mathfrak{OeisId}$ that elaborates to an IMMT constant declaration representing the sequence (as a function from integers to values).
2. The start values lines as well as example lines are both represented using the subfeature $\mathfrak{Example}$ introduced for STEX in Section 7.2. Specifically, the starting values are considered as examples of sequence elements.
3. For the name, reference and author lines we do not use features as there is no real structure or sensible elaboration. Instead they are represented as metadata using the Dublin Core standard.
4. For the formula lines we use an unary subfeature $\mathfrak{OeisForm}$ whose argument E is the term representing the parsed formula. Then, the elaboration function $\mathcal{E}$ to IMMT constant whose type is $\Vdash E$. Effectively each formula is an proposition about the sequence symbol so we represent it as an assertion stating that the proposition holds (hence using $\Vdash E$).

The OEIS formula format OEIS formulas are written in an ASCII-inspired syntax but there is no standardized grammar for it. However, in practice the language is regular enough that a partial parser and interpreter is realizable. The formula parser is outside the scope of this thesis and we refer to [LIK15] for details. Instead, the relevant parts for this document is that the result of the parser is a partially formal expression mixing content and narrative aspects. Therefore, we use the same approach for representing formulas as for STEX formulas described above in 7.2.

Implementation The OEIS importer is implemented as an IMMT extension that produces an IMMT document from each OEIS file. The code is available at `https://github.com/UniFormal/MMT/tree/master/src/mmt-oeis`. It contains around 2500 lines of Scala

code and 60 classes but that includes the formula parser which is not directly part of the work described here. The result of the import is available online at `https://gl.mathhub.info/oeis/oeis` and amounts to 740MB for the 250000 OEIS files.

Chapter 8

Applications for Flexiformal Knowledge

As libraries of machine-processable mathematical knowledge (such as the ones discussed in the previous chapter) grow, so does the benefit of building knowledge-management applications and semantics services for them. But, so far management support for and semantic services deployed on such libraries have been essentially insular. Large libraries like the MML (Mizar Mathematical Library) come with some pre-built tools but existing cross-library methods remain experiments, and have not been implemented into usable systems. In particular, applications for formalized content are mostly limited to proof-checking. This limits the value of mechanizing mathematics and, therefore, restricts the growth of associated systems, libraries and communities. In fact, as mentioned in Chapter 1, in [Wie07b], Freek Wiedijk identified the lack of a "killer application" as one of the major reasons for the failure of the QED project.

In this chapter we look at building such services and applications on top of the IMMT system based on the libraries we imported in the previous chapter. In the process we evaluate both the IMMT language and system with respect to their suitability for designing and implementing such tools. In a sense, the current formal representation languages for mathematics contribute to this problem as their machine-oriented, semantics-focused design makes them less suitable for building user-facing applications. This is the problem we tackled in the design of the IMMT language by focusing on being able to adequately represent (and reference) the narrative structure of mathematical knowledge alongside its semantics. Therefore, the IMMT language allows representing (and referencing via URIs) at least some of the mathematical constructs that mathematicians write about or think with. This means that within the IMMT system we can explore a wider range of human-oriented potential applications for (possibly partially) formalized mathematical knowledge. While we do not claim to have developed a "killer application" as part of this thesis, we do a substantial case study from the services and application-building perspective to evaluate the IMMT language and system. Within our case study we distinguish two distinct application domains with a different but correlated set of problems.

First we analyze the perspective of the library developer in the context of a heterogeneous, integrated flexiformal library. Even though there is one deep language where knowledge is integrated (i.e. IMMT in our case) there are multiple *surface syntaxes* in which content is written (e.g. STEX,

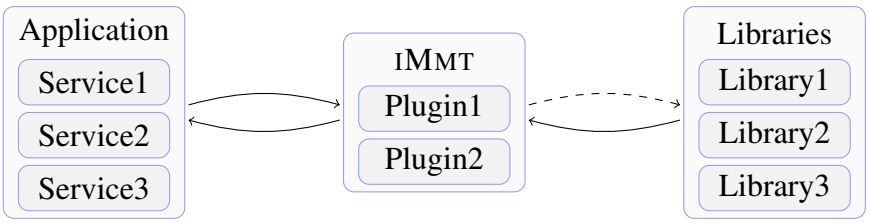


Figure 8.1: Basic Architecture of an IMMT-based Application

Mizar, etc.). This means that, in order to realize cross-library support in practice, a generic solution for integration, compilation, and error reporting must be realized within a universal *build system* for flexiformal libraries.

Second we explore the perspective of library users who are looking to find and read knowledge encoded within. But, underneath there are multiple source libraries describing and relating different concepts, focusing on distinct aspects of the mathematical knowledge and encoding it in various surface languages. For example the MML focuses on formalizing high-school and college-level mathematics while SMGloM has a broader scope but focuses on the narrative definitions in multiple languages. Therefore, even *accessing knowledge* in this heterogeneous, multi-format context is non-trivial. In order to build practical, user-facing applications for browsing and searching IMMT libraries, we need to realize a generic, uniform gateway for accessing IMMT knowledge.

Figure 8.1 shows the central components of a prototypical IMMT-based application. The libraries as a whole represent a *data store* containing the source files and the IMMT documents generated by the imports and, therefore, integrating multiple libraries written in different surface languages. Then, the IMMT system functions as a *semantic database* that makes the data available uniformly to services via a variety of queries. It may additionally write back to the data store, for example if used within authoring application. The application is the container that the user interacts with and it often integrates a series of different but related services. The services query IMMT as needed to provide the desired functionality. IMMT *plugins* can be used to extend IMMT queries and results where needed by particular services. For example, a web-based application that has a service for browsing IMMT documents, may implement (and use) a particular HTML-presenter as an IMMT plugin.

This chapter is organized as follows. First, in Section 8.1 we discuss the IMMT system as a semantic database enabling services to access and process the knowledge encoded in the libraries. Then, in Section 8.2 we present the MathHub system a generic IMMT-based application container that integrates several different applications and a wide variety of web-based services.

8.1 The IMMT System as a Semantic Database

In Chapter 7 we discussed imports for various libraries into IMMT. Specifically, we imported the Mizar Mathematical Library (Section 7.1), the SMGloM and GenCS libraries (Section 7.2) and,partially, the OEIS library (Section 7.3). Additionally, we also have the LATIN library [CHK+11] as a native MMT library in IMMT. LATIN (Logic Atlas and Integrator) is a relatively small, highly-modular library containing logics, foundations (e.g. Mizar) and basic mathematical concepts (such as numbers and algebraic structures).

By importing and integrating these libraries we effectively create one, heterogeneous, flexiformal mathematical library in IMMT. This is due to the genericity of IMMT which permits representing content that is heterogeneous not only with respect to the source language but also the level of formality. Then, applications can leverage the fact that, even though the libraries are independent, they often describe the same mathematical concepts. For example, the concept of *group* is defined in Mizar (see [GpM]), in LATIN (see [GpL]), as well as in SMGloM in both English (see [GpSa]) and German (see [GpSb]). These definitions represent different *facets* of the mathematical concept (for more discussion on this see [KKMR16]). The Mizar and LATIN ones are different formalizations in distinct logics while the SMGloM ones are narrations in different languages.

Therefore, the integrated IMMT library we build through imports in the previous chapter is genuinely flexiformal and provides a unique knowledge base on top which to build practical, user-facing applications. From the perspective of applications and services the IMMT system is a gateway to this knowledge encoded in IMMT libraries. As discussed above, this is essential because uniform access to the underlying knowledge is necessary for building cross-library applications. In practice, such access can only be provided at the IMMT level due to the heterogeneity of the libraries and surface languages.

In this regard, the basic functionality of the IMMT system is providing direct access based on IMMT URIs. However, it can be extended using *plugins* to answer complex queries or process the result in complex ways such as presenting it into particular formats (e.g. HTML). We discuss this in more detail in Section 8.1.1 below.

But the IMMT system can do more, by providing access to knowledge that is not explicitly represented in the libraries but that can be inferred through arbitrarily complex reasoning steps. We call this *induced material* and discuss below in Section 8.1.2 how the IMMT system can generate it and then make it accessible to services and applications.

8.1.1 Accessing Explicitly Represented Content

The IMMT system implements the IMMT language (by extending the MMT system) and provides an API for accessing and processing its knowledge base. The basic mechanism is retrieving knowledge items by using their IMMT URI but more complex queries and processing are possible via IMMT plugins. This is where the constraint discussed in Section 5.2.2 that each declaration,

including narrative ones, must have its own URI is especially relevant. It allows building user-facing applications that work with those (typically narrative) constructs that humans understand and are familiar with.

URI-based Retrieval Having an URI for each knowledge item is an MMT invariant which we preserve in IMMT. Therefore, any item can be retrieved by the IMMT system given its IMMT URI, assuming it is valid. The service making use of URI-based retrieval can further process the item before returning it to its user or client application. For example *definition lookup* is a basic application using URI-based retrieval. Typically, it additionally *presents* the retrieved definition with a given presenter depending on the context in which it is used. We discuss definition lookup as a service in Section 8.2.2 below.

Querying Relational Knowledge MMT provides the QMT [Rab12] query language which is implemented as a query engine in the MMT system. Since the IMMT system is build by extending the MMT system and because we preserve the central MMT invariants (most importantly the existence of URIs) we can make use of QMT naturally in IMMT. This enables services and applications to build complex queries involving standard unary and binary relations between URIs.

As part of this thesis, we extend the QMT implementation by adding *relational extractor* as new type of IMMT plugin. Instances of this plugin can be loaded at startup and can be associated with IMMT theories such as logical frameworks or foundations. They add new kinds of relations to the IMMT ontology and implement methods to extract their instances from imported knowledge. Therefore, by declaring the right relational extractor, we can build applications that make use of domain-specific queries and relations. For example, the SMGloM-specific "`isTranslationOf`" or "`isHypernymOf`" relations between concepts can be added to the ontology and extracted during importing. Relational extractors allow language designers in IMMT to extend the IMMT ontology of relations and use that for their applications. Therefore, they achieve for relational information, what structural features achieve for language constructs: enabling a more adequate representation of the underlying knowledge.

Then, we can enhance existing applications such as browsing the libraries with new relation-based services such as adding interactive links, information about related concepts or dedicated graph-based presenters for visualizing relational information. Additionally, we can build applications such as a dictionary using the translation relation or a thesaurus using the hypernymy (and synonymy) relations. We discuss concrete services in Section 8.2.2.

8.1.2 Accessing Induced Material

Acknowledgement. *Part of the results in this Section are based on collaborative work with Michael Kohlhase and have been published in [IK15b].*

As mentioned in Chapter 1, *math literate* MKM services are those that make use of the entire *space of mathematical knowledge* rather than just the part that is explicitly represented. We consider the mathematical knowledge space (MKS) as the knowledge that can be *induced* from the explicit one by using arbitrarily complex inference methods. Then, to build math literate MKM services we need to be able to generate (part of) the MKS within iMMT and make that available to services and applications.

In the context of theory graphs, we model the process of generating the knowledge space as an operation on theory graphs. Specifically, one that takes a theory graph Θ and returns an enriched graph $\overline{\Theta}$ where a new part of the mathematical knowledge space is explicitly represented. We call $\overline{\Theta}$ the *induced theory graph*.

Once the induced theory graph is generated, it can be made available by iMMT in the same way as explicit knowledge. This includes for instance relational information and exporting content. The key is that applications are able to work with the part of the knowledge space that is relevant for them, regardless of what (or how) it is explicitly represented. Additionally, a key aspect of iMMT (and MMT) is that its URI language is expressive enough to produce URIs for the induced statements that are not only unique but also informative. Specifically, we can compute the induced knowledge items from the induced theory graph by their URI and the original graph alone. Furthermore, we can generate explanations for the existence of each induced statement in terms of the original theory graph. We call this property of generated iMMT URIs *information completeness*.

Below we present two potential applications that can make use of induced knowledge. In the first case we model elaboration of derived declarations as generating part of the knowledge space. We apply that to *realms*, where realm faces represent the part of the knowledge space most relevant for practitioners or developers. In the second case, we generate the part of the knowledge space induced by views, which carry over defined concepts and theorems from their source to their target theory. In Section 8.2.2 we show a concrete implementation of a MATHWEBSEARCH-based search engine that searches statements induced by views in a highly modular library.

Inducing Realm Faces

The concept of *realms* in the context of theory graphs was introduced in [CFK14] and, in Section 6.1.4, we already briefly described realms and framed them in terms of elaborating structural features and derived declarations. Additionally, we gave a concrete algorithm for generating realm faces as induced by the underlying pillars (in [CFK14] they were meant to be manually produced and curated).

Realms were motivated by the intuition that users of a knowledge collection can have different roles and therefore want to see different kinds of materials. For instance, practitioners only need access to the face that supplies all the useful facts about a mathematical domain, whereas the librarian only wants to know how these are established and needs to access the "development history" in each of the pillars. Finally, the developer wants to develop the knowledge about the domain by extending one (or more) pillar, and the development of the face is just regarded as a

side effect.

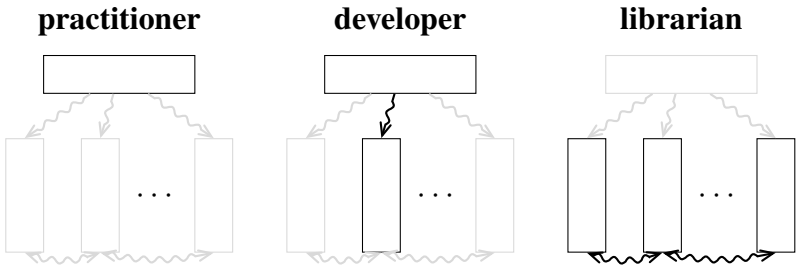


Figure 8.2: Realms and User Roles

In Figure 8.2 we visualize these idea by graying out the parts of the realms the users in their respective roles will not be able to see. We see each of the restricted theory graphs as the relevant *part of the knowledge space* for that particular class of users: *i*) the face graph – the faces inherit the graph structure of the developments – for the practitioners, *ii*) the graph of faces with selectively opened developments for the developer *iii*) the original theory graph for the knowledge librarian – a "user" of the library who maintains the library, e.g. refactoring theories, renaming theorems to avoid name clashes, etc.

In this section, we model this perspective using theory graphs and our interpretation of realms as structural features. The induced theory graphs for the practitioner and librarian case are defined below in Definition 28.

Definition 28. Given a theory graph Θ, the knowledge space of the *practitioner* is a theory graph $\overline{\Theta_p}$ that contains for every realm r in Θ a theory $r/face$ containing the declarations from the elaboration of r. Conversely, the knowledge space of the *librarian* is a theory graph containing for each realm r in Θ, every theory and view referenced in any of the pillars of r.

For the developer view described above we need the operation of *opening a pillar*. It allows the developer to access the "implementation details" (i.e. definiens) of the symbols and axioms in the face as formalized in one of the pillars. We model this by extending the elaboration function for realms. We create a new theory for each pillar that combines the symbol aggregation and name abstraction of the face theory with the implementation details of that pillar.

Concretely, given a realm `r` and a pillar `p` we induce a theory `r/p` that is generated following the same procedure as the face (i.e. from Definition 21) but without omitting the definitions. For symbols in a different pillar we generate the definition by translating it over the view from that pillar into `p`. Effectively, we obtain the symbol definitions *as seen from pillar* `p` which corresponds to the intuition of opening a pillar.

In the theory graph, we represent this as 1. a structure `s` from the pillar `p` that adds its symbols to `r/p` but with the renamings used in the face generation and, 2. a view `v` : `r` $\to$ `r/p` that formalizes the relation that `r/p` is an implementation for the face.

$$
\begin{array}{l}
\text{theory } \texttt{group/circ} = \{ \\
\quad \texttt{G} : \texttt{type}, \circ : \texttt{G} \to \texttt{G} \to \texttt{G}, \texttt{e} : \texttt{G}, \texttt{inv} : \texttt{G} \to \texttt{G}, \\
\quad / : \texttt{G} \to \texttt{G} \to \texttt{G} = \lambda a, b.a \circ (\texttt{inv}\ b), \\
\quad \texttt{circ/inv\_thm} : \Vdash\ (\texttt{inv e}) \doteq \texttt{e} = \texttt{trans}\ \texttt{e}_{\texttt{right}}\ \texttt{inv}_{\texttt{ax}}, \\
\quad \texttt{slash/inv\_thm} : \Vdash\ (\texttt{inv e}) \doteq \texttt{e} = \texttt{e}_{\texttt{ax}_2}\ \texttt{e}, \\
\quad \vdots \\
\}
\end{array}
$$

Figure 8.3: Developer View Example

Figure 8.3 shows the theory group/circ representing the opening of pillar circ in the realm of groups introduced above in Section 6.1.4. It has the same symbols as the face but with all symbols that are non-primitive in circ having a definition. For the two symbols originating in the second pillar ($/$ and slash/inv_thm) their definition is obtained by translating over $\texttt{v}_\circ$.

The resulting theory graph is shown in Figure 8.4 where the content of each theory is omitted. Note that the theories group, group/circ, group/slash as well as the structures $\texttt{s}_\circ$ and $\texttt{s}_/$ and the views $\texttt{i}_\circ$, $\texttt{i}_/$, $\texttt{f}_\circ$ and $\texttt{f}_/$ are all induced.

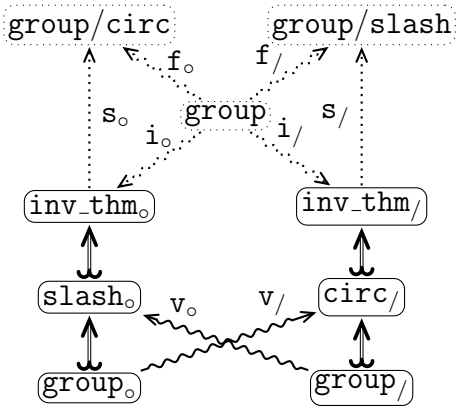


Figure 8.4: Opening a Pillar

Induced Knowledge using Views

Below, we define theory graph *flattening* as an instance of generating (part of) the implicit knowledge space. Specifically, it produces those statements induced by framing which we model with IMMT (i.e. MMT) views.

Definition 29. Given a theory graph Θ the flattening of a theory T in Θ is a theory $\overline{\texttt{T}}$ with the same URI as T containing:

- all constant declarations that are in T.
- all constant declarations that are imported into T.
- for every view $\texttt{v} : \texttt{S} \to \texttt{T}$ the projection of every S-based declaration over view v. Here, by S-based declaration we refer to the declarations in S and in conservative extensions of S.

Note that checking whether a theory T importing a theory S is actually a conservative extension of S is not straightforward in general. We omit a discussion here but note that in some particular

cases it can be done automatically and in other cases (e.g. the SMGloM library) it is done manually by the library authors.

The URIs of the induced declarations are based on the definition of IMMT URIs from Section 5.2.2 and permit recovering the origin of the induced declarations (i.e. are *information complete* as defined above. Specifically, constant declarations from $\mathtt{T}$ or imported in $\mathtt{T}$ by an include preserve their name. Meanwhile constant declarations induced by a view are additionally namespaced with the name of that view (in the same way as some of the statements from Figure 8.3 are namespaced with the pillar names).

Now we can define flattening of a theory graph based on the flattening of a theory defined above.

Definition 30. The flattening of diagram Θ is a diagram $\overline{\Theta}$ with the same URI as Θ containing the following module declarations:

- for every theory $\mathtt{T}$ in Θ the theory $\overline{\mathtt{T}}$
- a copy of every view $\mathtt{v} : \mathtt{S} \to \mathtt{T}$ from Θ

Note that, since theory flattening preserves theory URIs and does not add new axioms (only new theorems), every view $\mathtt{v}$ in Θ is also a view in $\overline{\Theta}$ so flattening can be iterated (infinitely many times if the theory graph has cycles) but the induced knowledge becomes less and less useful with every iteration.

8.2 The MathHub System

Acknowledgement. *Part of the results in this section are based on collaborative work with Constantin Jucovschi, Michael Kohlhase, and Tom Wiesing and have been published in [IJKW14]. However, we focus here only on the work done by the author.*

Motivated by our work with building and importing libraries as well as, more generally, the relative lack of applications for mechanized mathematics, we built the MathHub system with a dual goal. First, it aims to provide a basis for our own library development efforts. This includes supporting the contribution of new content to internally-developed libraries (e.g. SMGloM) but also the development and testing of importers for external libraries (e.g. MML) and providing a platform for archiving and publishing the result. Second, the MathHub system is a generic container that supports developing and deploying IMMT-based semantic services and applications. Concretely, in addition to implementing common aspects of most applications (such as user management) it provides an API for interacting with the IMMT system and an abstraction layer for IMMT archives and documents. Then, user-facing services and applications can be rapidly prototyped on top and deployed either as experiments or as practical tools for knowledge management.

Therefore, the MathHub system is a case study for the flexiformal vision by building a standalone ecosystem of flexiformal knowledge, services and applications. At the same time, it evaluates the IMMT language and system both from the perspective of representing content (libraries) and from the perspective of building applications.

8.2.1 System Architecture and Realization

MathHub is realized as an instance of the Drupal-based Planetary System [Koh12], which we have substantially extended in the course of the work reported here.

The system architecture has three main components: *i*) a versioned *data store* holding the source documents *ii*) a *semantic service provider* that imports the source documents and provides services for them *iii*) and a *frontend* that makes the sources and the semantic services available to users. Specifically, we use the GitLab repository manager [Git] as the data store, the IMMT API as the semantic service provider and Drupal as the frontend.

Figure 8.5 shows the detailed architecture.

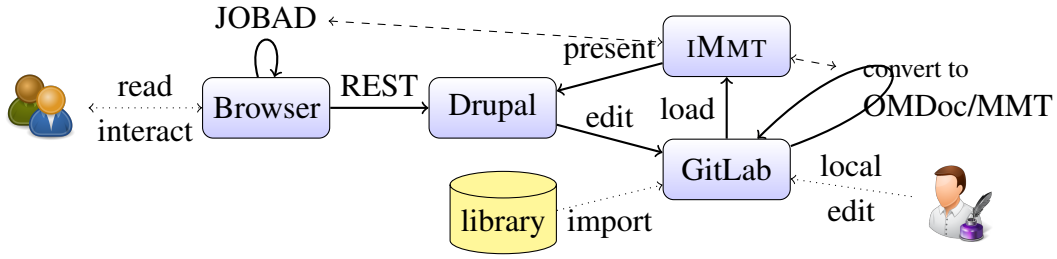


Figure 8.5: The MathHub Architecture

In this setup, Drupal serves as a container management system[1] that supplies uniform theming, user management, discussion forums, etc. GitLab on the other hand, provides versioned storage of the content documents, and organizes them into repositories owned by users and groups. On the user side, we use the JavaScript library JOBAD [GLR09, KGLZ09] to enhance the HTML delivered to the browser where needed.

Building an application for MathHub typically involves three implementation steps:

1. an IMMT plugin that processes the underlying knowledge and answers the queries as needed by the application

2. a Drupal module that interacts with IMMT (using the APIs provided by MathHub in Drupal) and builds the page(s), assembles the response(s) from IMMT, and renders the result as HTML

[1]Drupal and similar systems self-describe as content management systems, but they actually only manage the documents without being aware their internal structure.

3. a JOBAD module that provides in-browser interactivity and which may query IMMT itself as a response to user actions.

A concrete deployment of the MathHub system can be instantiated by providing (using a configuration file) the IMMT archives that should be available, the location (URL) of the associated IMMT instance, and a list of (identifiers of) implemented applications that should be enabled. The most interesting part here, is the abstractions for the archives which facilitate application building and support library development within MathHub.

8.2.2 Services and Applications

In this section we present several concrete examples of applications and services built on top of the IMMT API. Most are developed as part of the MathHub system except for the search systems which are standalone applications. However, these are also loosely integrated into MathHub via links.

Browsing Documents

The ability to browse the documents from the data store in a uniform interface is one of the basic applications of MathHub. As mentioned above, this is non-trivial because the content from the data store may come from various libraries, be written in different languages and, finally, be presented to the user using different IMMT exporters. Therefore, in order to provide such an uniform interface we need to build a generic application that abstracts over the libraries and their formats as well as the importers or exporters that are used to process the documents.

Concretely, we describe each archive by a set of *formats* for the kind of source languages it contains. Typically this is one format such as `mizar` for the Mizar Mathematical Library (MML) or `stex` for the two sTEX libraries: SMGloM and GenCS. But archives can use additional formats such as auxiliary files for organization or documentation. For instance, we consider folders used for organizing knowledge within an archive as an additional such format `folder`. Then, each folder represents an IMMT document consisting of document references for each file inside the folder.

Formats themselves are characterized by a set of *compilers* which define how content in that format will be processed by the IMMT system. Compilers are split into two categories, *importers* and *exporters*. The former process source documents to produce, possibly in multiple stages, IMMT content. The latter process IMMT content to produce documents that are consumed by external applications such as web browsers (`html`) or image viewers (`svg`). Each format is therefore associated with a chain of importers and a set of exporters. For example the `mizar` format has two importers: one into `xmlabs` XML documents and the second from `xmlabs` into IMMT. The exporters are mostly format-independent as they work on already-imported IMMT content. Rather, they are application-dependent so the exporters are determined by the requirements of the desired services and applications.

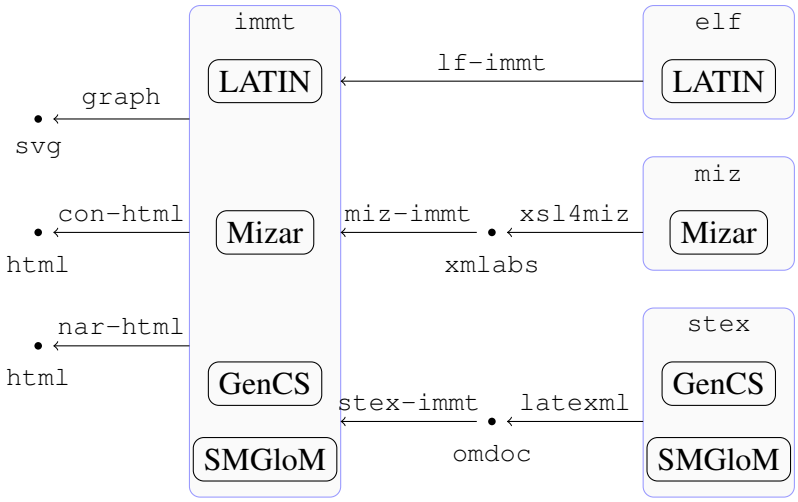


Figure 8.6: Compiler Chain for IMMT Archives

Figure 8.6 shows the compilers and formats for the main IMMT archives. Conceptually, each processing stage represents the same IMMT document with a different and possibly partial serialization. Therefore, we consider each intermediate format as a different *aspect* for one IMMT document. For instance, for STEX files, there is a source aspect (in `stex`) and a presentation aspect (in `html`) as well as an OMDOC aspect for the result of `latexml` processing.

Based on these abstractions we implement a *crawler* in MathHub that traverses archives and creates the corresponding MathHub documents for each relevant file in the data store. The MathHub system then automatically aggregates the aspects of each document and presents them as tabs in the web page allowing users to easily cycle between them. For instance, the example in Figure 8.7 shows an SMGloM document for which the aspects are the HTML presentation (default), the SVG graph of dependencies, the result of the OMDOC compiler, and, finally, the original STEX source. Therefore, not only the content in each tab but also the tabs themselves are generic and are instantiated for each format using the configuration described above. This allows MathHub to realize the uniform interface for browsing despite the heterogeneity of the source documents.

Additionally, we enhance the HTML-based browsing interface with services for interactivity. We build interactive documents by using a context menu in the web page that provides access to specific services which we discuss individually below. As mentioned above, we use the JOBAD JavaScript library for the context menu and interactivity.

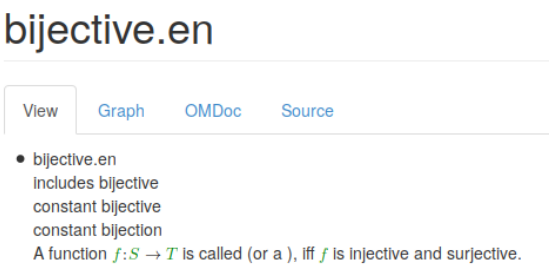


Figure 8.7: Rendering Aspects in MathHub

To realize each of the services we implement two things. First we write a `ServerPlugin` for the IMMT API that implements the server-side

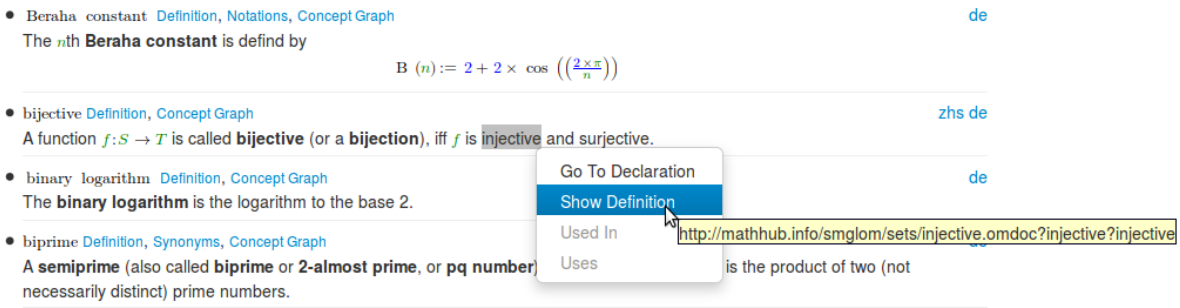


Figure 8.8: Definition Lookup in MathHub

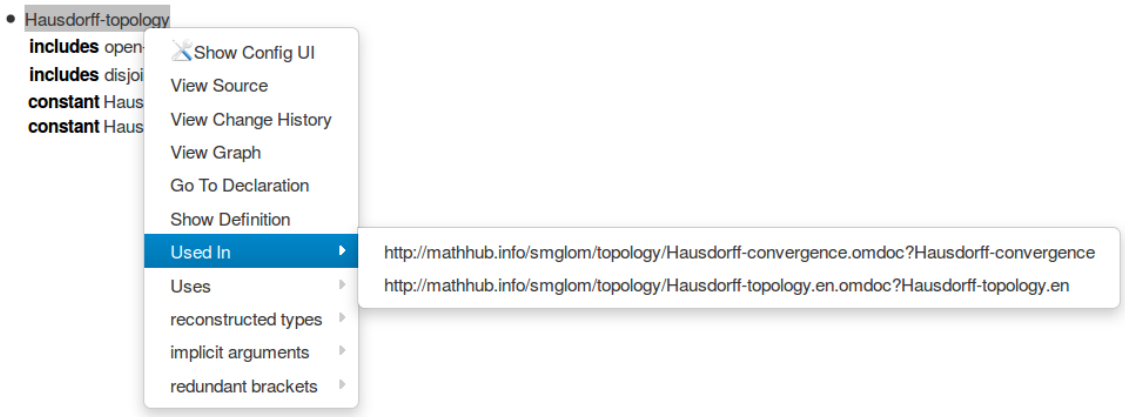


Figure 8.9: Relational-based Navigation

functionality. Then we write a JOBAD module that adds the relevant context menu entries and then interfaces with the IMMT server to retrieve the result and render it for the user. The code implementing the services on the IMMT side is available at [IDi]. The corresponding JOBAD modules are available at [IDJ].

For the IMMT API, *definition lookup* (shown in Figure 8.8) is a basic service using URI-based retrieval. But, it also uses domain-specific relational information. For example, in SMGloM, due to the multilingual design there are multiple definitions which may each be in a different document or theory. Additionally, IMMT presents the retrieved definition with a given presenter.

Another example of a basic service is *relational navigation* which uses the relational index and the QMT query language [Rab12]. For instance, Figure 8.9 shows the relational information integrated into the context menu, that can be used to navigate the relational graph.

Authoring Content

In terms of authoring content MathHub builds upon existing services provided by the IMMT system and the Drupal framework. Concretely, IMMT provides a semi-automatic *build system* for processing (importing and exporting) documents and Drupal provides basic editing func-

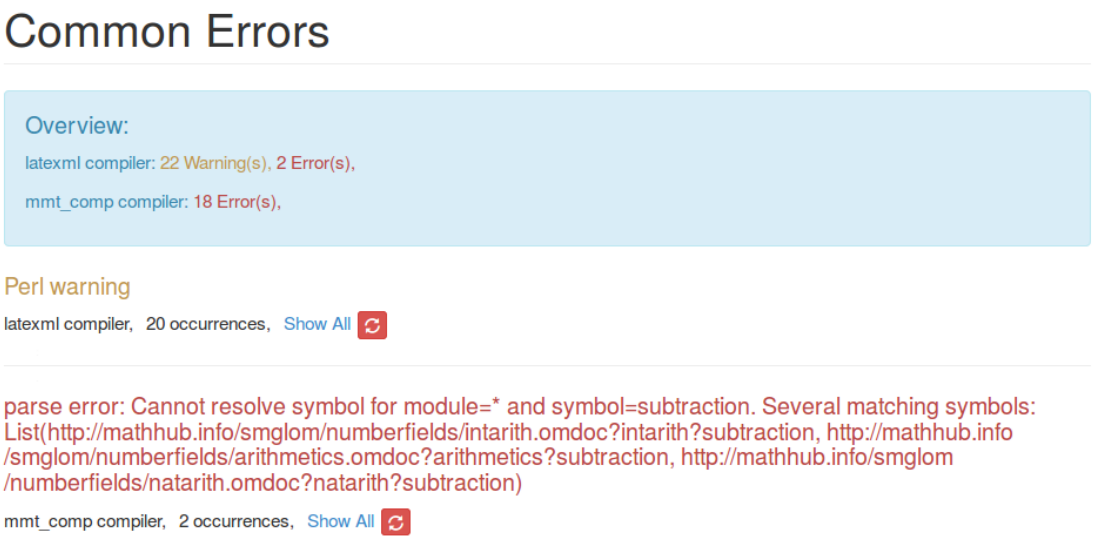


Figure 8.10: Aggregating Errors in MathHub

tionality. Effectively, we develop a basic and experimental web IDE (Integrated Development Environment) for developing IMMT knowledge on the web. Note that MathHub also supports offline authoring by integrating with the *LMH* tool [LMH15] which is more practical for bulk editing jobs (however LMH is outside the scope of this thesis).

At this point, the *editing* support is basic and minimally extends pre-existing Drupal functionality. Features such as IMMT-powered generic syntax highlighting, auto-complete, type inference or semantic high-lighting remain future work.

For the *build system* we develop an administration interface that allows updating the sources and libraries as well as rebuilding selected documents or archives by sending them to the build queue. Additionally, we produce, parse and store error information and make it available in the interface. We aggregate common errors in a dedicated page (see Figure 8.10) where developers can filter and inspect common errors for specific compilers and libraries.

We also show errors as annotations to the source of each document. Figure 8.11 shows an example sTEX document from the GenCS library defining abstract data types. The source references are carried over and used to align errors on the sidebar and highlight the location where they occur according to the compiler. Clicking on the error opens the editor line in question and highlights the relevant text region. The sidebar infrastructure that we use is part of the JOBAD library.

The MathHub build system was heavily and routinely used in developing the SMGloM library and improving related sTEX tools. However, most of the editing was done offline in bulk jobs and deployed regularly to MathHub.

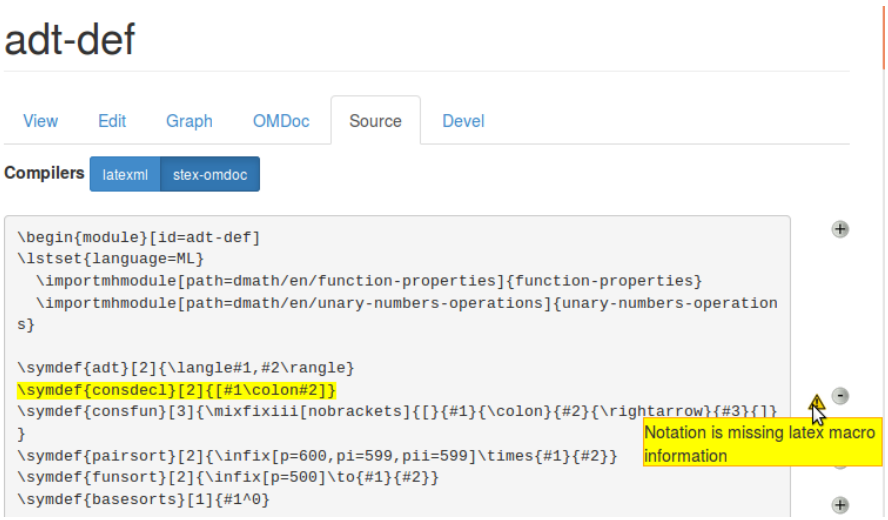


Figure 8.11: Error Annotations in MathHub

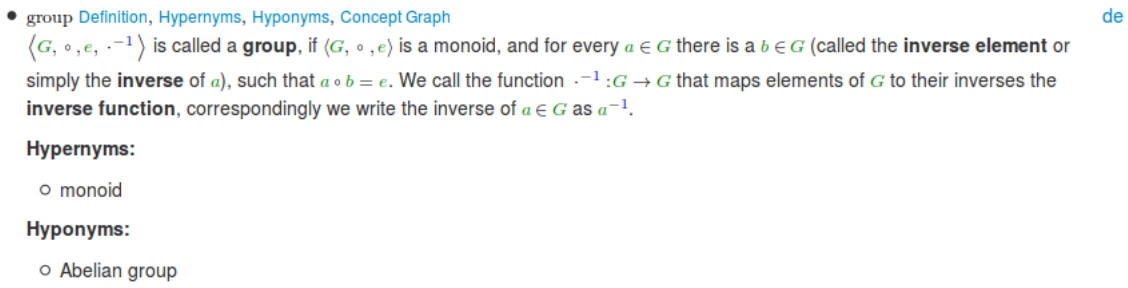


Figure 8.12: An Entry in the SMGloM Thesaurus

Mathematical Thesaurus and Dictionary

As part of developing the SMGloM library, we built two applications that make specific use of the kind of knowledge encoded in it. The SMGloM (Semantic Multilingual Glossary of Mathematics) library has mathematical terms together with narrative definitions and notations in multiple languages (typically English and German). Based on that we built a mathematical *thesaurus* and a *dictionary*.

Figure 8.12 shows an entry in the SMGloM thesaurus where the definition is annotated with links to expand its definition, concept graph or terminological relations such as hyper- and hypo-nymy. The relations are extracted from the sTEX sources during the import by looking at metadata encoding this information. The sTEX importer comes with an associated relational extractor that declares the `isHypernym` and `isHyponym` binary relations and implements the extraction algorithm. Then, the thesaurus application makes use of the relations during presentation. Additionally, on the right, the entry links to the corresponding definitions in other languages are shown (in this particular case only the German version).

Figure 8.13 shows the dictionary application built on top of SMGloM knowledge. It uses the

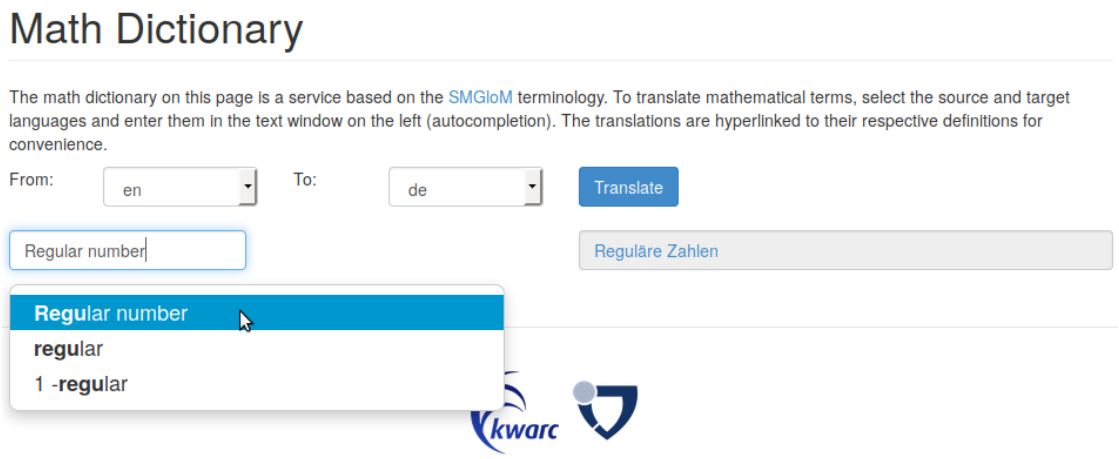


Figure 8.13: The SMGloM Dictionary

verbalization notations of the concepts together with the language associated with each of them to do the translation. The interface is straightforward and a basic auto-complete function based on available verbalizations is implemented for the input. A similar approach can be used to provide auto-complete while authoring IMMT content online in MathHub but, as mentioned above, this remains future work.

Note that the particular applications presented here (as well as the library on which they are based) are just examples. The key insight here is that MathHub and IMMT are generic enough to allow building and deploying such domain-specific applications that leverage the particularities of the original knowledge bases. Therefore, they provides a platform for future development and experimenting with application design.

Text and Formula Search

Being able to adequately search large corpora of mathematical knowledge requires being able to construct queries that involve both text and formulas. But, the problem of mathematical information retrieval is an important and complex, standalone research question that is outside the scope of this thesis. Therefore, we do not attempt to solve that here as a standalone IMMT plugin and refer to [AKOS14] for a recent overview of systems for mathematical information retrieval. Instead, we take an existing solution (the MATHWEBSEARCH system – discussed below) and integrate it with IMMT.

MathWebSearch At its core, the MATHWEBSEARCH system [PK11, KMP12] is a content-oriented search engine for mathematical expressions. It indexes formula-URL pairs and provides a web interface for querying the formula index via unification. In [Pro14] it was extended with a text component to allow search for both textual and formula queries at the same time. In that case the frontend has two text fields for both the text and formula components of the queries.

Figure 8.14: Text and Math Search for SMGloM and GenCS

Then, the query components and results are aggregated by the MATHWEBSEARCH system. The input for MATHWEBSEARCH is a MATHWEBSEARCH *harvest* which is a set of XHTML files containing text and formulas as MATHML. The files need to contain specific metadata such as author and title and the formulas need to have unique identifiers and their content MATHML representation (not just presentation MATHML).

Therefore, to realize the integration between IMMT and MATHWEBSEARCH, we wrote an exporter from IMMT content to MATHWEBSEARCH harvests. The code is integrated into the IMMT API and is available at [MEx]. Based on that we can develop concrete instances of the MATHWEBSEARCH system that allow querying IMMT knowledge through the MATHWEBSEARCH interface.

Concretely, as part of the MathHub experiment we have developed a reusable deployment of the MATHWEBSEARCH infrastracture integrated with the IMMT system. It is realized as a Docker [Mer14] build and installation script that can be used to instantiate concrete search applications. The code is available at `https://github.com/KWARC/TEMA_docker` and the specific instances that were built and deployed are available as docker virtual machines at `https://hub.docker.com/r/kwarc/tema_search/`. Within the MathHub project we have created and deployed two such systems.

The first is a search instance for the LaTeX and sTeX based libraries in the MathHub system, in particular the SMGloM and the GenCS libraries. An example query is shown in Figure 8.14 and the demo is available at `http://jupiter.eecs.jacobs-university.de/mathhub/`.

The second instance is for the OEIS library and, improving on the search system already provided by the OEIS community, it also allows searching for formulae. An example query is shown in

Figure 8.15: Text and Math Search for the OEIS library

Figure 8.15 and the demo is available at `http://oeissearch.mathweb.org/`.

MATHWEBSEARCH exemplifies how pre-existing, standalone applications can be loosely integrated with MathHub. The index used by MATHWEBSEARCH is produced by IMMT once, then the MATHWEBSEARCH service can load it and answer queries. The MATHWEBSEARCH interface is integrated in MathHub only as a link but uniform theming makes the transition seamless.

Math Literate Search

Acknowledgement. *Part of the results in this Section are based on collaborative work with Michael Kohlhase and Corneliu Prodescu and have been published in [IKP14]. However we only focus here on the work done by the author.*

As a first application of the concepts described above in Section 8.1.2 we build a system for searching the knowledge space (i.e. flattened diagram) of the highly modular LATIN [CHK+11] library.

Induced statements in the LATIN library LATIN (Logic Atlas and Integrator) is written in an extension of the Twelf implementation [RS09] of LF [HHP93]. So it is natural to use an extension of LF notation with query variables as the input language for the query formula. Therefore, we use the IMMT notation language and interpretation service described in [IR12b] to transform LF-style input into IMMT objects and subsequently to MWS queries.

We implemented library flattening for inducing knowledge via views as described in Section 8.1.2 in IMMT and applied it to the LATIN library. The flattening (one iteration) of the LATIN library increases the number of declarations from 2 310 to 58 847 (a factor of 25.4) and the total size of the library from 123.9 MB to 1.8 GB (a factor of 14.8). As expected, the multiplication factor depends on the level of modularity of the library. For instance, the highly modular math sub-library containing mainly algebraic structures increases from 2.3 MB to 79 MB thus having a multiplication factor of 34.3, more than double the library average. The size of the MWS harvests also increases considerably, from 25.2 MB to 539.0 MB.

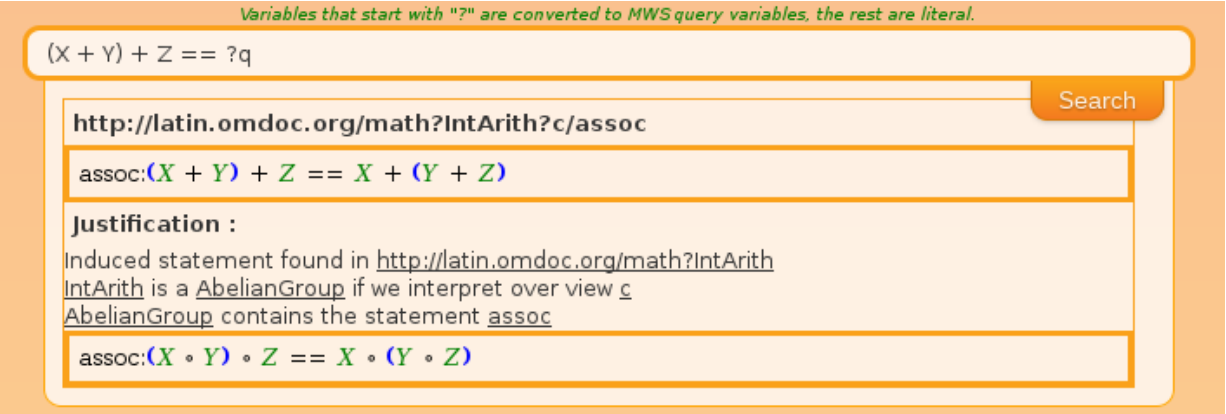


Figure 8.16: The FLATSEARCH Web Interface for LATIN

Explaining the origin of induced statements The induced constants are produced by translating a constant c in a theory (or sub-theory of) $\mathtt{S}$ over a view $\mathtt{v} : \mathtt{S} \to \mathtt{T}$. This induces a new constant in T with the intuition that it is a constant from (the sub-theory of) $\mathtt{S}$ as seen when interpreting via view $\mathtt{v}$. We encode this origin of the new constant by using the name of the view as part of its local name. Therefore the new constant has local name $\mathtt{v/c}$ and therefore, global name (i.e. IMMT URI) $\mathtt{T?v/c}$. Note that, for the URI $\mathtt{T?v/c}$ to be valid the view URI v is string-escaped to be a valid local name fragment and then un-escaped by the application when needed. This means that, when presenting the induced constant in the application we can use its IMMT URI to produce a human-understandable justification for its existence. The explanation is generated from the IMMT URI by a straightforward template-based algorithm following the intuition above.

The resulting search interface is shown in Figure 8.16 and is available at [FS]. In the example query from Figure 8.16, the associativity of addition is induced from the theory `AbelianGroup` when interpreting over a view that maps $\circ$ to $+$. The application inspects the induced IMMT URI as discussed above to produce the justification shown in the figure.

8.2.3 Implementation Overview

The libraries, systems and applications described above come together to form an integrated management system for flexiformal knowledge. Figure 8.17 shows an overview of the software components that were implemented or extended. For each concrete service or system, we use colors to mark the cases where only part of the implementation was done as part of this thesis.

In the case of library imports, the importer framework was already in IMMT and, as mentioned in Section 7.3, the formula parser of the OEIS importer was implemented separately [LIK15]. Moreover, while the authoring and search *applications* described here were implemented as part of MathHub, they rely on pre-existing systems to do the heavy-lifting. In the case of search that is

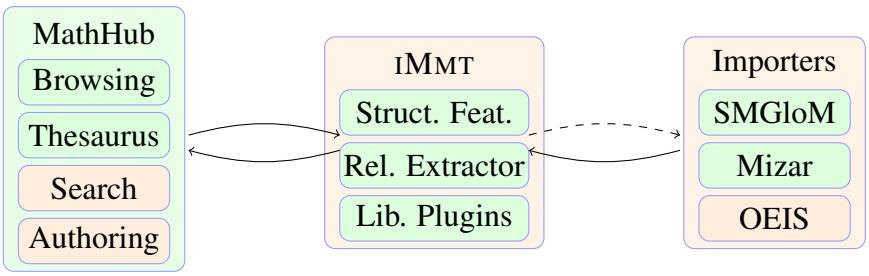


Figure 8.17: The MathHub Ecosystem

the MATHWEBSEARCH system, while for authoring it is the Drupal content management system. Finally, the IMMT system is, as described in previous Chapters, an extension of the MMT system. We extended it with support for structural features, the relational extractor described above as well as various library-specific plugins (e.g. web-server plugins) used by the MathHub services.

A list of the individual implementations and extensions with links to the respective code repositories can be found in [Imp]. In particular, the code for the MathHub system is available at `https://github.com/KWARC/MathHub` and a Docker configuration for easily deploying new MathHub instances is available at `https://github.com/KWARC/MathHub_docker`. The extensions we contributed to the code as part of this thesis amount to around 4500 lines of code. There are currently four instances of MathHub that are publicly deployed. One is restricted to the SMGloM libraries and project (deployed at `smglom.mathhub.info`), the second is restricted to the ODK project [ODK, DIK+16] and related libraries (deployed at `odk.mathhub.info`). The third is restricted to the fully formal libraries with the OAF project [OAF] (deployed at `oaf.mathhub.info`) and, finally, the main MathHub instance is deployed at `http://mathhub.info` and has reached a state where it can be used for experiments and accessing its resources. It currently has around 10 000 documents and a couple dozen users and repositories.

Therefore, we consider it a significant case study for developing flexiformal mathematical content as well as services and applications based on it. Concretely, the currently deployed libaries are: *i*) the SMGloM termbase with ca. 1500 small STEX files containing definitions of mathematical terminology and notation definitions. *ii*) ca. 6500 files with STEX-encoded teaching materials (slides, course notes, problems, and solutions) in Computer Science, *iii*) the LATIN logic atlas with ca. 1000 meta-theories and logic morphisms, *iv*) the Mizar Mathematical Library of ca. 1000 articles with ca. 50.000 theorems, definitions, and proofs, and *v*) a part of the HOL Light Library with 22 theories and over 2800 declarations.

While the implementation itself still an experiment, the MathHub system is already unique in the sense that it gives a unified interface to multiple mathematical libraries and provides a platform for using, building and deploying applications and services for them.

Chapter 9

Conclusion and Future Work

Meaningful mechanization of mathematics requires most mathematical knowledge to be represented in machine-understandable formats. A current limitation is the effort required for converting mathematical knowledge into such formats, either by natural language processing or by formalization. Moreover, most current formats focus on one aspect of mathematics, be it either narration, computation or formal semantics. But mathematical practice leverages all of these aspects together, so focusing on just one reduces the range of potential applications and, therefore, decreases the incentive for producing such machine-understandable knowledge. We identified *flexiformality* [KK11, Koh13] as a potential solution to alleviate both of these problems, by lowering the cost of converting mathematical knowledge, and by co-representing narration and semantics to expand the space of potential applications. In this thesis, we have conducted an initial, comprehensive case study in flexiformality and explored the three objectives that we established in Chapter 1.

The first was *representing flexiformal knowledge* and we started by surveying existing formal and informal mathematics in Chapter 2. We observed that narration and structure are consequential components of mathematical knowledge and, therefore, essential for adequately representing mathematics. Moreover, we identified a set of flexiformal phenomena that occur in practice which together function as a specification for a flexiformal representation language. Then, in Chapter 5 we introduced the iMMT language and system as an experimental solution by extending the MMT language and system with a narrative aspect and an extensible declaration level. Finally, in Chapter 6 we evaluated it with respect to this flexiformal specification by describing how to represent each of the outstanding flexiformal phenomena from Chapter 2 in iMMT.

Second, we addressed *flexiformalization* in Chapter 7 by importing existing mathematical libraries into iMMT as a generic, uniform, flexiformal representation format. The main result is establishing a large, flexiformal knowledge base to enable future experiments in flexiformalization. We chose representative libraries at different levels of formality, to show that the iMMT language and system are suitable in practice for representing the entire spectrum of flexiformal knowledge. The libraries in question, the Mizar Mathematical Library [TB85, MML] (fully formal), the SMGloM [GIJ+15] (and GenCS) library (semi-formal), and the OEIS library [OEI15]

(informal) are each among the largest in their class. Therefore, importing them into the same representation language and effectively creating a large, heterogeneous, flexiformal library, is a major step forward on its own.

Third, in Chapter 8, we tackled *flexiformal knowledge management* by designing and implementing the MathHub system as a generic framework for building and deploying practical user-facing applications. Moreover, we explored the breath of potential applications by implementing and integrating several concrete instances for browsing, authoring and searching IMMT knowledge. In the process we validated the library imports as forming a valuable knowledge base as well as the central design ideas of the IMMT language. In particular, we observed that the ability of IMMT to represent and reference the narrative constructs that human users understand was essential for each of these applications. As mentioned in Section 1, Freek Wiedijk identified in [Wie07b] the lack of a "killer application" as one the major reasons for the failure of the QED project, alongside the small size of the formalized mathematics community. But, in the MathHub system, library development is incentivized by integration with applications that can make immediate use of additional knowledge. Therefore, while still an experiment, MathHub aims to become a driving force for future development of both applications and libraries.

But, as mentioned above, the work presented in this thesis represents only an initial case study and the first step towards flexiformal mathematics. Within the language design objective, future work includes a more in-depth study of structural features and representing their translation at the pragmatic level within views. Moreover, a more practical and scalable solution for underspecification will be needed for applying IMMT to represent, on a large scale, alternative parses of complex expressions. Additionally we have not yet fixed a concrete surface syntax for IMMT– since we, instead, relied on imports to collect mathematical knowledge. Some work in this direction has been started in [IKRY] by mixing the MMT and STEX syntaxes but it is still work in progress.

With respect to libraries, many remain to be imported, in particular large informal archives partially formalized by natural language processors. Moreover, within the already imported ones more integration of their content is desirable. Along this direction the work in [KKMR16] may prove fruitful.

On the system side, more concrete services and applications are still to be developed. In particular a major current shortcoming is the lack of a powerful semantic editor for IMMT content in MathHub. One approach to solve this is to scale the backend of the existing jEdit-based editor for MMT [Rab14a] to IMMT and the web. An implementation for this is currently underway within the IMMT system. Also, generally, the evaluation of applications presented here was from a developer perspective, exploring what kind of services can be built effectively. But a comprehensive case study from a users perspective, focusing on which flexiformal services and applications are practically useful for mathematicians remains future work.

Bibliography

[ABC+10] Ron Ausbrooks, Stephen Buswell, David Carlisle, Giorgi Chavchanidze, Stéphane Dalmas, Stan Devitt, Angel Diaz, Sam Dooley, Roger Hunter, Patrick Ion, Michael Kohlhase, Azzeddine Lazrek, Paul Libbrecht, Bruce Miller, Robert Miner, Murray Sargent, Bruce Smith, Neil Soiffer, Robert Sutor, and Stephen Watt. Mathematical Markup Language (MathML) version 3.0. W3C Recommendation, World Wide Web Consortium (W3C), 2010.

[ACR+08] Serge Autexier, John Campbell, Julio Rubio, Volker Sorge, Masakazu Suzuki, and Freek Wiedijk, editors. *Intelligent Computer Mathematics*, number 5144 in LNAI. Springer Verlag, 2008.

[AFP] Archive of formal proofs. `https://www.isa-afp.org/`. seen June 2016.

[AKOS14] Akiko Aizawa, Michael Kohlhase, Iadh Ounis, and Moritz Schubotz. NTCIR-11 Math-2 task overview. In Noriko Kando, Hideo Joho, and Kazuaki Kishida, editors, *NTCIR 11 Conference*, pages 88–98, Tokyo, Japan, 2014. NII, Tokyo.

[And02] Peter B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*. Springer, second edition, 2002.

[APS] Art of Problem Solving. `https://www.artofproblemsolving.com/community/c13_contests`. seen July 2016.

[ARCT11] Andrea Asperti, Wilmer Ricciotti, Claudio Sacerdoti Coen, and Enrico Tassi. The matita interactive theorem prover. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 64–69. Springer, 2011.

[ArX] `arxiv.org` e-Print archive. `http://www.arxiv.org`. seen June 2016.

[ASC06] Serge Autexier and Claudio Sacerdoti Coen. A formal correspondence between omdoc with alternative proofs and the $\overline{\lambda}\mu\tilde{\mu}$-calculus. In Jon Borwein and William M. Farmer, editors, *Mathematical Knowledge Management (MKM)*, number 4108 in LNAI, pages 67–81. Springer Verlag, 2006.

BIBLIOGRAPHY

[Avi08] Jeremy Avigad. Understanding proofs. In Paolo Mancosu, editor, *The Philosophy of Mathematical Practice*, pages 317–353. Oxford University Press, 2008.

[Ban03] Grzegorz Bancerek. On the structure of Mizar types. *Electronic Notes in Theoretical Computer Science*, 85(7), 2003.

[Bar15] Jeremiah Bartz. Induced and Complete Multinets. *ArXiv e-prints*, February 2015.

[Bau99] Judith Baur. Syntax und semantik mathematischer texte – ein prototyp. Master's thesis, Fachrichtung Computerlinguistik, Universität des Saarlandes, Saarbrücken, Germany, 1999.

[BC04] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development — Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer Verlag, 2004.

[BCC+04] Stephen Buswell, Olga Caprotti, David P. Carlisle, Michael C. Dewar, Marc Gaëtano, and Michael Kohlhase. The Open Math standard, version 2.0. Technical report, The OpenMath Society, 2004.

[BDN09] Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda - A functional language with dependent types. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, 22nd International Conference, TPHOLs 2009, Munich, Germany, August 17-20, 2009. Proceedings*, volume 5674 of *Lecture Notes in Computer Science*, pages 73–78. Springer, 2009.

[Ber91] Paul Bernays. *Axiomatic Set Theory*. Dover Publications, 1991.

[Bou74] Nicolas Bourbaki. *Algebra I*. Elements of Mathematics. Springer Verlag, 1974.

[BP64] Paul Benacerraf and Hilary Putnam, editors. *Philosophy of mathematics: Selected readings*. Cambridge University Press, 2nd edition 1983 edition, 1964.

[CAL+13] Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors. *Intelligent Computer Mathematics*, number 7961 in Lecture Notes in Computer Science. Springer, 2013.

[CCK09] Merlin Carl, Marcos Cramer, and Daniel Kühlwein. Chapter 1 from Landau in Naproche. `https://korpora-exp.zim.uni-duisburg-essen.de/naproche/downloads/2009/landauChapter1.pdf`, 2009.

[CDSCW09] Jacques Carette, Lucas Dixon, Claudio Sacerdoti Coen, and Stephen M. Watt, editors. *MKM/Calculemus Proceedings*, number 5625 in LNAI. Springer Verlag, July 2009.

BIBLIOGRAPHY

[CF08] Jacques Carette and William M. Farmer. High-level theories. In Autexier et al. [ACR+08], pages 232–245.

[CFK+10] Marcos Cramer, Bernhard Fisseni, Peter Koepke, Daniel Kühlwein, Bernhard Schröder, and Jip Veldman. The Naproche project controlled natural language proof checking of mathematical texts. In Norbert E. Fuchs, editor, *Controlled Natural Language, Workshop on Controlled Natural Language, CNL 2009. Revised Papers*, number 5972 in LNCS, pages 170–186. Springer, 2010.

[CFK14] Jacques Carette, William Farmer, and Michael Kohlhase. Realms: A structure for consolidating knowledge about mathematical theories. In Watt et al. [WDS+14], pages 252–266. MKM Best-Paper-Award.

[CH88] Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, 1988.

[CHK+11] Mihai Codescu, Fulya Horozal, Michael Kohlhase, Till Mossakowski, and Florian Rabe. Project abstract: Logic atlas and integrator (LATIN). In Davenport et al. [DFRU11], pages 289–291.

[Chr03] Jacek Chrzaszcz. Implementing modules in the Coq system. In David A. Basin and Burkhart Wolff, editors, *Theorem Proving in Higher Order Logics, 16th International Conference, TPHOLs 2003, Rom, Italy, September 8-12, 2003, Proceedings*, volume 2758 of *Lecture Notes in Computer Science*, pages 270–286. Springer, 2003.

[CIM14] David Carlisle, Patrick Ion, and Robert Miner. Guidelines for graphics in MathML 2. W3C Note, 2014.

[CMR16] Mihai Codescu, Till Mossakowski, and Florian Rabe. Selecting colimits for parameterisation and networks of specifications. In *Workshop on Algebraic Development Techniques*, 2016.

[Coh81] Paul M. Cohn. *Universal Algebra*. D. Reidel Publishing, revised edition, 1981.

[Coq96] Projet Coq. The Coq proof assistant (version 6.0) — reference manual. Technical report, ENS Lyon – INRIA Rocquencourt, 1996.

[CS09] Cris Calude and Ludwig Staiger. A note on accelerated turing machines. CDMTCS Research Report 350, Centre for Discrete Mathematics and Theoretical Computer Science, Auckland University, 2009.

[dB87] Nicolaas Govert de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In P. Dybjer et al., editors, *Proceedings of the Workshop on Programming Languages*, 1987.

BIBLIOGRAPHY

[DFRU11] James Davenport, William Farmer, Florian Rabe, and Josef Urban, editors. *Intelligent Computer Mathematics*, number 6824 in LNAI. Springer Verlag, 2011.

[DIK+16] Paul-Olivier Dehaye, Mihnea Iancu, Michael Kohlhase, Alexander Konovalov, Samuel Lelièvre, Dennis Müller, Markus Pfeiffer, Florian Rabe, Nicolas M. Thiéry, and Tom Wiesing. Interoperability in the opendreamkit project: The math-in-the-middle approach. In Michael Kohlhase, Moa Johansson, Bruce R. Miller, Leonardo de Moura, and Frank Wm. Tompa, editors, *Intelligent Computer Mathematics - 9th International Conference, CICM 2016, Bialystok, Poland, July 25-29, 2016, Proceedings*, volume 9791 of *Lecture Notes in Computer Science*, pages 117–131. Springer, 2016.

[dMKA+15] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 378–388. Springer, 2015.

[DT00] Marc Dymetman and Frédéric Tendeau. Context-free grammar rewriting and the transfer of packed linguistic representations. In *COLING 2000, 18th International Conference on Computational Linguistics*. Morgan Kaufmann, 2000.

[EFT94] Heinz-Dieter Ebbinghaus, Jörg Flum, and Wolfgang Thomas. *Mathematical Logic*. Springer, second edition, 1994.

[EuD] The european digital mathematics library. `https://eudml.org/`. seen July 2016.

[Far07] William M. Farmer. Biform theories in Chiron. In Manuel Kauers, Manfred Kerber, Robert Miner, and Wolfgang Windsteiger, editors, *Towards Mechanized Mathematical Assistants. MKM/Calculemus*, number 4573 in LNAI, pages 66–79. Springer Verlag, 2007.

[Far08] William M. Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6(3):267–286, September 2008.

[FGT92] William M. Farmer, Josuah Guttman, and Xavier Thayer. Little theories. In D. Kapur, editor, *Proceedings of the 11th Conference on Automated Deduction*, number 607 in LNCS, pages 467–581, Saratoga Springs, NY, USA, 1992. Springer Verlag.

[FGT93] William M. Farmer, Joshua D. Guttman, and F. Javier Thayer. IMPS: An Interactive Mathematical Proof System. *Journal of Automated Reasoning*, 11(2):213–248, October 1993.

BIBLIOGRAPHY

[FS] FlatSearch demo. http://cds.omdoc.org:8181/search.html. seen July 2016.

[GAA+13] Georges Gonthier, Andrea Asperti, Jeremy Avigad, Yves Bertot, Cyril Cohen, Fran/cois Garillot, Stéphane Le Roux, Assia Mahboubi, Russell O'Connor, Sidi Ould Biha, Ioana Pasca, Laurence Rideau, Alexey Solovyev, Enrico Tassi, and Laurent Théry. A machine-checked proof of the odd order theorem. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings*, volume 7998 of *Lecture Notes in Computer Science*, pages 163–179. Springer, 2013.

[Gan13] Mohan Ganesalingam. *The Language of Mathematics, A Linguistic and Philosophical Investigation*, volume 7805 of *LNCS*. Springer Verlag, 2013.

[GAP16] GAP system for computational discrete algebra. http://www.gap-system.org/, 2016. seen August 2016.

[Gel04] Gijs Geleijnse. Comparing two user-friendly formal languages for mathematics: Weak Type Theory and Mizar. Master's thesis, Technische Universiteit Eindhoven, 2004.

[GIJ+15] Deyan Ginev, Mihnea Iancu, Constantin Jucovschi, Andrea Kohlhase, Michael Kohlhase, Heinz Kröger, Jürgen Schefter, and Wolfram Sperber. The SMGLoM project and system. 2015.

[GIJ+16] Deyan Ginev, Mihnea Iancu, Constantin Jucovshi, Andrea Kohlhase, Michael Kohlhase, Akbar Oripov, Jürgen Schefter, Wolfram Sperber, Olaf Teschke, and Tom Wiesing. The SMGloM project and system. towards a terminology and ontology for mathematics. In Gert-Martin Greuel, Thorsten Koch, Peter Paule, and Andrew Sommese, editors, *Mathematical Software - ICMS 2016 - 5th International Congress*, volume 9725 of *LNCS*. Springer, 2016.

[Gin11] Deyan Ginev. The structure of mathematical expressions. Master's thesis, Jacobs University Bremen, Bremen, Germany, August 2011.

[Git] GitLab system. https://about.gitlab.com/. seen July 2016.

[GLR09] Jana Giceva, Christoph Lange, and Florian Rabe. Integrating web services into active mathematical documents. In Carette et al. [CDSCW09], pages 279–293.

[GpL] Definition of "Group" in the LATIN logic atlas. https://gl.mathhub.info/MMT/LATIN/blob/master/source/math/magmas.elf#L173. seen August 2016.

BIBLIOGRAPHY

[GpM] Definition of "Group" in the Mizar Mathematical Library. http://www.mizar.org/version/current/html/group_1.html#NM1. seen August 2016.

[GpSa] "English" definition of "Group" in the semantic multilingual glossary of mathematics (SMGloM). https://gl.mathhub.info/smglom/algebra/blob/master/source/group.en.tex. seen August 2016.

[GpSb] "German" definition of "Group" in the semantic multilingual glossary of mathematics (SMGloM). https://gl.mathhub.info/smglom/algebra/blob/master/source/group.de.tex. seen August 2016.

[GS90] Jeroen Groenendijk and Martin Stokhof. Dynamic Montague Grammar. In L. Kálmán and L. Pólos, editors, *Papers from the Second Symposium on Logic and Language*, pages 3–48. Akadémiai Kiadó, Budapest, 1990.

[GS91] Jeroen Groenendijk and Martin Stokhof. Dynamic predicate logic. *Linguistics & Philosophy*, 14:39–100, 1991.

[HAB+15] Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the kepler conjecture. *CoRR*, abs/1501.02155, 2015.

[Har84] David Harel. Dynamic Logic. In D. M. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604. Reidel, Dordrecht, 1984.

[Har11] John Harrison. The HOL Light theorem prover. http://www.cl.cam.ac.uk/~jrh13/hol-light/, 2011. seen September 2016.

[HHP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. *Journal of the Association for Computing Machinery*, 40(1):143–184, 1993.

[HK15] Tapani Hyttinen and Kaisa Kangas. On model theory of covers of algebraically closed fields. http://arxiv.org/pdf/1502.01042.pdf, 2015. seen June 2016.

[HKR12] Fulya Horozal, Michael Kohlhase, and Florian Rabe. Extending MKM formats at the statement level. In Jeuring et al. [JCC+12], pages 65–80.

[IDi] iMMT code for interactive documents. https://github.com/UniFormal/MMT/blob/master/src/planetary-mmt/src/info/kwarc/mmt/planetary/ServerPlugin.scala. seen August 2016.

BIBLIOGRAPHY

[IDJ] JOBAD code for interactive documents. https://github.com/KWARC/MathHub/tree/master/sites/all/modules/mmt/jobad. seen August 2016.

[IJKW14] Mihnea Iancu, Constantin Jucovschi, Michael Kohlhase, and Tom Wiesing. System description: Mathhub.info. In Watt et al. [WDS+14], pages 431–434.

[IK15a] Mihnea Iancu and Michael Kohlhase. A flexiformal model of knowledge dissemination and aggregation in mathematics. In Kerber et al. [KCK+15], pages 137–152.

[IK15b] Mihnea Iancu and Michael Kohlhase. Math literate knowledge management via induced material. In Kerber et al. [KCK+15], pages 187–202.

[IKP14] Mihnea Iancu, Michael Kohlhase, and Corneliu-Claudiu Prodescu. Representing, archiving, and searching the space of mathematical knowledge. In Hoon Hong and Chee Yap, editors, *Mathematical Software - ICMS 2014 - 4th International Congress*, volume 8592 of *LNCS*, pages 26–30. Springer, 2014.

[IKR] Mihnea Iancu, Michael Kohlhase, and Florian Rabe. Understanding the pragmatics of module systems for mathematics.

[IKR11] Mihnea Iancu, Michael Kohlhase, and Florian Rabe. Translating the Mizar Mathematical Library into OMDoc format. KWARC report, Jacobs University Bremen, 2011.

[IKRU13] Mihnea Iancu, Michael Kohlhase, Florian Rabe, and Josef Urban. The Mizar Mathematical Library in OMDoc: Translation and applications. *Journal of Automated Reasoning*, 50(2):191–202, 2013.

[IKRY] Mihnea Iancu, Michael Kohlhase, Florian Rabe, and Hang Yuan. Mixing surface languages for OMDoc. http://kwarc.info/kohlhase/submit/mmt-stex16.pdf.

[Imp] Thesis implementation: Overview of systems and extensions. https://kwarc.info/miancu/papers/thesis-code.html. seen April 2017.

[IR98] Kenneth Ireland and Michael Rosen. *A Classical Introduction to Modern Number Theory*. Springer, second edition, 1998.

[IR11] Mihnea Iancu and Florian Rabe. Formalizing foundations of mathematics. *Mathematical Structures in Computer Science*, 21(4):883–911, 2011.

[IR12a] Mihnea Iancu and Florian Rabe. Management of change in declarative languages. In J. Campbell, J. Carette, G. Dos Reis, J. Jeuring, P. Sojka, V. Sorge, and M. Wenzel, editors, *Intelligent Computer Mathematics*, pages 325–340. Springer, 2012.

[IR12b] Mihnea Iancu and Florian Rabe. (work-in-progress) an MMT-based user-interface. In *Workshop on User Interfaces for Theorem Provers*, 2012.

[JCC+12] Johan Jeuring, John A. Campbell, Jacques Carette, Gabriel Dos Reis, Petr Sojka, Makarius Wenzel, and Volker Sorge, editors. *Intelligent Computer Mathematics*, number 7362 in LNAI. Springer Verlag, 2012.

[Joj05] Gueorgui I. Jojgov. Translating a fragment of weak type theory into type theory with open terms. In Michael Kohlhase, editor, *Mathematical Knowledge Management, MKM'05*, number 3863 in LNAI, pages 389–403. Springer Verlag, 2005.

[JOM] JOMDoc project — Java library for OMDoc documents. seen Aug. 2015.

[KCK+15] Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors. *Intelligent Computer Mathematics*, number 9150 in LNCS. Springer, 2015.

[Ker10] Manfred Kerber. Proofs, proofs, proofs, and proofs. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors, *Intelligent Computer Mathematics*, number 6167 in LNAI, pages 345–354. Springer Verlag, 2010.

[KGLZ09] Michael Kohlhase, Jana Giceva, Christoph Lange, and Vyacheslav Zholudev. JOBAD – interactive mathematical documents. In Brigitte Endres-Niggemeyer, Valentin Zacharias, and Pascal Hitzler, editors, *AI Mashup Challenge at KI Conference*, September 2009.

[KI14] Michael Kohlhase and Mihnea Iancu. Co-representing structure and meaning of mathematical documents. *Sprache und Datenverarbeitung, International Journal for Language Data Processing*, 38(2):49–80, 2014. Special Issue "The language of mathematics – computational, linguistic and logical aspects".

[KK09] Andrea Kohlhase and Michael Kohlhase. Spreadsheet interaction with frames: Exploring a mathematical practice. In Carette et al. [CDSCW09], pages 341–356.

[KK11] Andrea Kohlhase and Michael Kohlhase. Towards a flexible notion of document context. In *Proceedings of the 29th annual ACM international conference on Design of communication (SIGDOC)*, pages 181–188, New York, NY, USA, 2011. ACM Special Interest Group for Design of Communication, ACM Press.

[KKMR16] Cezary Kaliszyk, Michael Kohlhase, Dennis Müller, and Florian Rabe. A standard for aligning mathematical concepts. In *Intelligent Computer Mathematics – Work in Progress Papers*, 2016.

[KKP96] Michael Kohlhase, Susanna Kuschert, and Manfred Pinkal. A type-theoretic semantics for λ-DRT. In P. Dekker and M. Stokhof, editors, *Proceedings of the 10th Amsterdam Colloquium*, pages 479–498, Amsterdam, 1996. ILLC.

[KMP12] Michael Kohlhase, Bogdan A. Matican, and Corneliu C. Prodescu. MathWebSearch 0.5 – Scaling an Open Formula Search Engine. In Jeuring et al. [JCC+12], pages 342–357.

[KMR08] Michael Kohlhase, Christine Müller, and Florian Rabe. Notations for living mathematical documents. In S. Autexier, J. Campbell, J. Rubio, V. Sorge, M. Suzuki, and F. Wiedijk, editors, *Mathematical Knowledge Management*, pages 504–519. Springer, 2008.

[KMR13] Michael Kohlhase, Felix Mance, and Florian Rabe. A universal machine for biform theory graphs. In Carette et al. [CAL+13].

[KMW04] Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Mathlang: Experience-driven development of a new mathematical language. *Electr. Notes Theor. Comput. Sci.*, 93:138–160, 2004.

[KN04] Fairouz Kamareddine and Rob Nederpelt. A refinement of de Bruijn's formal language of mathematics. *Logic, Language and Information*, 13(3):287–340, 2004.

[Koh06] Michael Kohlhase. *OMDoc – An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, August 2006.

[Koh08] Michael Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2(2):279–304, 2008.

[Koh10] Michael Kohlhase. OMDoc version 1.3. `http://omdoc.org/OMDoc1.3`, 2010.

[Koh12] Michael Kohlhase. The Planetary project: Towards eMath3.0. In Jeuring et al. [JCC+12], pages 448–452.

[Koh13] Michael Kohlhase. The flexiformalist manifesto. In Andrei Voronkov, Viorel Negru, Tetsuo Ida, Tudor Jebelean, Dana Petcu, Stephen M. Watt, and Daniela Zaharie, editors, *14th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*, pages 30–36, Timisoara, Romania, 2013. IEEE Press.

[Koh14a] Michael Kohlhase. A data model and encoding for a semantic, multilingual terminology of mathematics. In Watt et al. [WDS+14], pages 169–183.

[Koh14b] Michael Kohlhase. Mathematical knowledge management: Transcending the one-brain-barrier with theory graphs. *EMS Newsletter*, pages 22–27, June 2014.

[Koh15] Michael Kohlhase. Modular documents, document URIs and late binding. `https://gl.kwarc.info/omdoc/blue/blob/master/docuris/note.pdf`, 2015. KWARC Blue Note.

[Koh16] Michael Kohlhase. sTeX: Semantic markup in TEX/LATEX. Technical report, 2016.

[KR93] Hans Kamp and Uwe Reyle. *From Discourse to Logic: Introduction to Model-Theoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer, Dordrecht, 1993.

[KRSC11] Michael Kohlhase, Florian Rabe, and Claudio Sacerdoti Coen. A foundational view on integration problems. In Davenport et al. [DFRU11], pages 107–122.

[KTP10] Alexander Koller, Stefan Thater, and Manfred Pinkal. Scope underspecification with tree descriptions: Theory and practice. In Matthew Crocker and Jörg Siekmann, editors, *Resource Adaptive Cognitive Processes*, Cognitive Technologies Series. Springer, Berlin, 2010.

[KW99] Florian Kammuller and Markus Wenzel. Locales: A sectioning concept for Isabelle. In *Theorem Proving in Higher Order Logics (TPHOLs 99)*, number 1690 in LNCS, pages 149–165. Springer, 1999.

[KW08] Fairouz Kamareddine and J. B. Wells. Computerizing mathematical text with MathLang. *Electr. Notes Theor. Comput. Sci.*, 205:5–30, 2008.

[Lan30] Edmund Landau. *Grundlagen der Analysis*. Wissenschaftliche Buchgesellschaft, Darmstadt, Germany; second edition, 1930. Reprint of the edition, Leipzig, 1970.

[Lan68] Serge Lang. *Analysis I*. Addison-Wesley, world student series edition, 1968.

[Lan10] Serge Lang. *Undergraduate Algebra*. Springer, third edition, 2010.

[LIK15] Enxhell Luzhnica, Mihnea Iancu, and Michael Kohlhase. Importing the OEIS library into OMDoc. In Ralph Bergmann, Sebastian Görg, and Gilbert Müller, editors, *Proceedings of the LWA 2015 Workshops: KDML, FGWM, IR, and FGDB*, pages 296–303, October 2015.

[LMH15] The LocalMH system. `https://github.com/KWARC/localmh`, 2015.

[Mel93] Erica Melis. Analogy between proofs – a case study. SEKI-Report SR-93-13, Fachbereich Informatik, Unversität des Saarlandes, 1993.

[Mer14] Dirk Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), March 2014.

[MEx] Code for MathWebSearch exporter in iMMT. `https://github.com/UniFormal/MMT/blob/master/src/mmt-api/src/main/info/kwarc/mmt/api/archives/MWSHarvestExporter.scala`. seen August 2016.

[Mil] Bruce Miller. `LaTeXML`: A LATEX to XML converter. Web Manual at `http://dlmf.nist.gov/LaTeXML/`. seen September 2011.

BIBLIOGRAPHY

[Miz] Mizar system. http://mizar.org/system/. seen June 2016.

[MML] Mizar Mathematical Library. http://www.mizar.org/library. seen June 2016.

[Mon74] Richard Montague. The proper treatment of quantification in ordinary English. In R. Thomason, editor, *Formal Philosophy. Selected Papers*. Yale University Press, New Haven, 1974.

[MPD16] MuPAD. http://mathworks.com/discovery/mupad.html, 2016. seen July 2016.

[Mpl16] Maplesoft. http://www.maplesoft.com/, 2016. seen July 2016.

[MRv] Mathematical reviews. http://www.ams.org/mr-database. seen July 2016.

[MS62] Jan Mycielski and Hugo Steinhaus. A mathematical axiom contradicting the axiom of choice. *Bulletin de l'Acadmie Polonaise des Sciences. Srie des Sciences Mathmatiques, Astronomiques et Physiques*, 1962.

[MWd] Wolfram MathWorld. http://mathworld.wolfram.com/. seen July 2016.

[Nei73] Neil J. A. Sloane. *A Handbook of Integer Sequences*. Academic Press, 1973.

[Nei94] Neil J. A. Sloane. An on-line version of the encyclopedia of integer sequences. http://www3.combinatorics.org/Volume_1/PDF/v1i1f1.pdf, 1994.

[Nei12] Neil J. A. Sloane. The on-line encyclopedia of integer sequences. http://neilsloane.com/doc/eger.pdf, 2012.

[Nel97] Roger B. Nelsen. *Proofs without Words: Exercises in Visual Thinking*. The Mathematical Association of America, 1997.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Number 2283 in LNCS. Springer, 2002.

[NS95] Neil J. A. Sloane and Simon Plouffe. *The Encyclopedia of Integer Sequences*. Academic Press, 1995.

[OAF] The OAF project & system. http://oaf.mathhub.info.

[ODK] OpenDreamKit open digital research environment toolkit for the advancement of mathematics. http://opendreamkit.org.

[Odl95] A. M. Odlyzko. Tragic loss or good riddance? the impending demise of traditional scholarly journals. *International Journal of Human-Computer Studies*, 42:71–122, 1995.

[OEI] OEIS help. `http://oeis.org/eishelp1.html`. seen June 2016.

[OEI15] OEIS Foundation Inc. The on-line encyclopedia of integer sequences. `http://oeis.org/`, 2015.

[Pau94] Lawrence C. Paulson. *Isabelle: A Generic Theorem Prover*. Number 828 in LNCS. Springer Verlag, 1994.

[Pau05] Lawrence C. Paulson. Isabelle reference manual. Technical report, Computer Laboratory, University of Cambridge, October 2005.

[Pfe91] Frank Pfenning. Logic programming in the LF logical framework. In Gérard P. Huet and Gordon D. Plotkin, editors, *Logical Frameworks*. Cambridge University Press, 1991.

[PK11] Corneliu C. Prodescu and Michael Kohlhase. Mathwebsearch 0.5 - open formula search engine. In *Wissens- und Erfahrungsmanagement LWA (Lernen, Wissensentdeckung und Adaptivität) Conference Proceedings*, September 2011.

[Pro14] Corneliu C. Prodescu. Text and formula search on ArXiv documents. M. sc. thesis, Jacobs University Bremen, 2014.

[Pus98] James Pustejovsky. The semantics of lexical underspecification. *Folia Linguistica*, 32:323–347, 1998.

[QED95] The QED manifesto. Internet Report `http://www.rbjones.com/rbjpub/logic/qedres00.htm`, 1995.

[QED96] The QED project. `http://www-unix.mcs.anl.gov/qed/`, 1996.

[Rab12] Florian Rabe. A Query Language for Formal Mathematical Libraries. In Jeuring et al. [JCC+12], pages 142–157.

[Rab13] Florian Rabe. The MMT API: A generic MKM system. In Carette et al. [CAL+13], pages 339–343.

[Rab14a] Florian Rabe. A Logic-Independent IDE. In C. Benzmüller and B. Woltzenlogel Paleo, editors, *Workshop on User Interfaces for Theorem Provers*, pages 48–60. Elsevier, 2014.

[Rab14b] Florian Rabe. MMT objects. In M. England, J. Davenport, A. Kohlhase, M. Kohlhase, P. Libbrecht, W. Neuper, P. Quaresma, A. Sexton, P. Sojka, J. Urban, and S. Watt, editors, *Workshops and Work in Progress at CICM 2014: OpenMath Workshop*. CEUR, 2014.

BIBLIOGRAPHY

[Rab15a] Florian Rabe. Generic literals. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, editors, *Intelligent Computer Mathematics*, pages 102–117. Springer, 2015.

[Rab15b] Florian Rabe. Lax theory morphisms. *ACM Transactions on Computational Logic*, 17(1), 2015.

[Rib96] Paulo Ribenboim. *The New Book of Prime Number Records*. Springer, third edition, 1996.

[RK13a] Florian Rabe and Michael Kohlhase. A scalable module system. *Information & Computation*, 0(230):1–54, 2013.

[RK13b] Marco Riccardi and Artur Kornilowicz. Fundamental group of n-sphere for $n \geq 2$. *Formalized Mathematics*, 20(2):97–104, 2013.

[RS09] Florian Rabe and Carsten Schürmann. A practical module system for LF. In J. Cheney and A. Felty, editors, *Proceedings of the Workshop on Logical Frameworks: Meta-Theory and Practice (LFMTP)*, volume LFMTP'09 of *ACM International Conference Proceeding Series*, pages 40–48. ACM Press, 2009.

[Rud73] Walter Rudin. *Functional Analysis*. McGraw Hill, 1973.

[Rud76] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, third edition, 1976.

[Sag16] SageMath. `http://www.sagemath.org/`, 2016. seen July 2016.

[Sca] The Scala programming language. `http://www.scala-lang.org`. seen July 2016.

[SK08] Heinrich Stamerjohanns and Michael Kohlhase. Transforming the arXiv to XML. In Autexier et al. [ACR+08], pages 574–582.

[SLP] sTeX plugin for LaTeXML. GitHub repository at `https://github.com/KWARC/LaTeXML-Plugin-sTeX`. seen July 2016.

[SMG] SMGloM: A semantic, multilingual terminology for mathematics. `http://smglom.mathhub.info`. seen August 2016.

[Spi94] Michael Spivak. *Calculus*. Cambridge University Press, third edition, 1994.

[sTe] sTeX: An infrastructure for semantic preloading of LaTeX documents. GitHub repository at `https://github.com/KWARC/sTeX`. seen July 2016.

[TB85] Andrzej Trybulec and Howard Blair. Computer Assisted Reasoning with Mizar. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, pages 26–28, 1985.

[TKUG13] Carst Tankink, Cezary Kaliszyk, Josef Urban, and Herman Geuvers. Formal mathematics on display: A wiki for Flyspeck. In Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors, *Intelligent Computer Mathematics - MKM, Calculemus, DML, and Systems and Projects 2013, Held as Part of CICM 2013, Bath, UK, July 8-12, 2013. Proceedings*, volume 7961 of *Lecture Notes in Computer Science*, pages 152–167. Springer, 2013.

[Try90] Andrzej Trybulec. Tarski Grothendieck set theory. *Formalized Mathematics*, 1(**1**):9–11, 1990.

[WBB+] Makarius Wenzel, Clemens Ballarin, Stefan Berghofer, Jasmin Blanchette, Timothy Bourke, Lukas Bulwahn, Amine Chaieb, Lucas Dixon, Florian Haftmann, Brian Huffman, Gerwin Klein, Alexander Krauss, Ondřej Kunčar, Tobias Nipkow, Lars Noschinski, David von Oheimb, Larry Paulson, Sebastian Skalberg, and Christian Sternagel. *The Isabelle/Isar Reference Manual*.

[WDS+14] Stephan Watt, James Davenport, Alan Sexton, Petr Sojka, and Josef Urban, editors. *Intelligent Computer Mathematics*, number 8543 in LNCS. Springer, 2014.

[WG10] Magdalena Wolska and Mihai Grigore. Symbol declarations in mathematical writing: A corpus study. In Petr Sojka, editor, *Towards Digital Mathematics Library, DML workshop*, pages 119–127. Masaryk University, Brno, 2010.

[WGK11] Magdalena Wolska, Mihai Grigore, and Michael Kohlhase. Using discourse context to interpret object-denoting mathematical expressions. In Petr Sojka, editor, *Towards Digital Mathematics Library, DML workshop*, pages 85–101. Masaryk University, Brno, 2011.

[Wie07a] Freek Wiedijk. Mizar's soft type system. In K. Schneider and J. Brandt, editors, *Theorem Proving in Higher Order Logics*, volume 4732 of *Lecture Notes in Computer Science*, pages 383–399. Springer, 2007.

[Wie07b] Freek Wiedijk. The QED manifesto revisited. *Studies in Logic, Grammar and Rhetoric*, 10(23):121–133, 2007.

[Wik] Wikipedia. `https://en.wikipedia.org/`. seen July 2016.

[Wol02] Stephen Wolfram. *The Mathematica Book*. Cambridge University Press, 2002.

[Wol12] Magdalena Wolska. Building a pos-annotated corpus of scientific papers in mathematics. In Petr Sojka and Michael Kohlhase, editors, *DML and MIR 2012*. Masaryk University, Brno, 2012. in press.

[Wol13] Magdalena Wolska. *Student's Language in Computer-Assisted Tutoring of Mathematical Proofs*. PhD thesis, ComputerLinguistik, Saarland University, 2013.

[ZBM] Zentrallblatt MATH. `https://zbmath.org/`. seen July 2016.

BIBLIOGRAPHY

[Zil03] Boris Zilber. Covers of the multiplicative group of an algebraically closed field of characteristic zero, 2003.

[Zin04] Claus Zinn. *Understanding Informal Mathematical Discourse*. PhD thesis, Technischen Fakultät der Universität Erlangen-Nürnberg, 2004.