The space of mathematical software systems

Jacques Carette, William M. Farmer, Yasmine Sharoda
Computing and Software
McMaster University
http://www.cas.mcmaster.ca/research/mathscheme/

Katja Berčič, Michael Kohlhase, Dennis Müller, Florian Rabe
Computer Science
FAU Erlangen-Nürnberg
http://kwarc.info

January 5, 2020

Abstract

Mathematical software systems are becoming more and more important in pure and applied mathematics in order to deal with the complexity and scalability issues inherent in mathematics. In the last decades we have seen a cambric explosion of increasingly powerful but also diverging systems.

To give researchers a guide to this space of systems, we devise a novel conceptualization of mathematical software that focuses on five aspects: *inference* covers formal logic and reasoning about mathematical statements via proofs and models, typically with strong emphasis on correctness; *computation* covers algorithms and software libraries for representing and manipulating mathematical objects, typically with strong emphasis on efficiency; *tabulation* covers generating and maintaining collections of mathematical objects conforming to a certain pattern, typically with strong emphasis on complete enumeration; *narration* covers describing mathematical contexts and relations, typically with strong emphasis on human readability; finally, *organization* covers representing mathematical contexts and objects in machine-actionable formal languages, typically with strong emphasis on expressivity and system interoperability.

Despite broad agreement that an ideal system would seamlessly integrate all these aspects, research has diversified into families of highly specialized systems focusing on a single aspect and possibly partially integrating others, each with their own communities, challenges, and successes. In this survey, we focus on the commonalities and differences of these systems from the perspective of a future multi-aspect system.

Our goal is to give new researchers, existing researchers from each of these communities, or outsiders like mathematicians a basic overview that enables them to match practical challenges to existing solutions, identify white spots in the software space, and to deepen the integration between systems and paradigms.

Contents

1 Introduction

In the last half decade we have seen mathematics tackle problems that lead to increasingly large developments: proofs, computations, data sets, and document collections. This trend has led to intense discussions about the nature of mathematics, ventilating questions like:

i) Is a proof that can only be verified with the help of a computer still a mathematical proof?
ii) Is a mathematical proofscape that exceeds what can be understood in detail by a single expert a legitimate justification of a mathematical result?
iii) Can a collection of mathematics papers — however big — adequately represent a large body of mathematical knowledge?

In [Car+19] we have discussed these questions under the heading of **Big Math** and propose a unified, high-level model. We claim that computer support will be necessary for scaling mathematics, and that suitable and acceptable methods should be developed in a tight collaboration between mathematicians and computer scientists — indeed such method development is already under way, but needs to become more comprehensive and integrative.

We propose that all Big Math developments comprise four main aspects that need to be dealt with at scale:

i) *Inference*: deriving statements by *deduction* (i.e., proving), *abduction* (i.e., conjecture formation from best explanations), and *induction* (i.e., conjecture formation from examples).
ii) *Computation*: algorithmic manipulation and simplification of mathematical expressions and other representations of mathematical objects.
iii) *Tabulation*: generating, collecting, maintaining, and accessing collections of examples that suggest patterns and relations and allow testing of conjectures.
iv) *Narration*: bringing the results into a form that can be digested by humans, usually in mathematical documents like articles, books, or preprints, that expose the ideas in natural language but also in diagrams, tables, and simulations.

These aspects — their existence and importance to mathematics — should be rather uncontroversial. Figure 1 may help convey the part which is less discussed, and not less crucial: that they are tightly related. For a convenient representation in three dimensions, we choose to locate the organization aspect at the barycentre of the other four since they are all consumers and producers of mathematical knowledge.

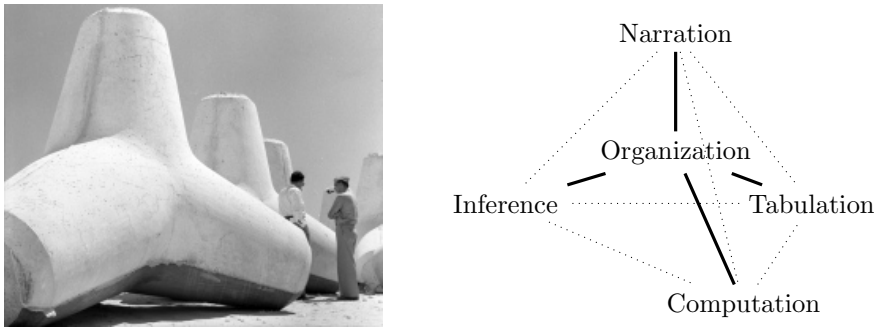

Figure 1: Five Aspects of Big Math Systems, a Tetrapod Structure

Computer support exists for all of these four aspects of Big Math, e.g.,

i) theorem provers like Isabelle, Coq, or Mizar;
ii) computer algebra systems like GAP, SageMath, Maple, or Mathematica; and
iii) mathematical data bases like the L-functions and Modular Forms Data Base (LMFDB) [Cre16; LM] and the Online Encyclopedia of Integer Sequences (OEIS) [Slo03];
iv) online journals, mathematical information systems like zbMATH or MathSciNet, preprint servers like arXiv.org, or research-level help systems like MathOverflow.

While humans can easily integrate these four aspects and do that for all mathematical developments (large or otherwise), much research is still necessary into how such an integration can be

achieved in software systems. [1] EdN:1

Overview We want to throw the spotlight on the integration problem to help start off research and development of systems that integrate all four aspects. To facilitate this, we give a high-level survey of mathematical software systems from the Tetrapod perspective. Because almost every one of these systems has one primary aspect and because systems with the same primary aspect are often very similar, we group systems by their primary aspects.

In each group we try to further subdivide the systems. We want to stress that this classification is mostly meant for convenience, e.g., by simplifying the description of several similar systems. We do not mean to imply a strict separation between these groups of systems, and often the borders are fluid. In particular, there are some systems that already allow aspect switching in a way that precludes ascribing a primary aspect. For convenience, we still assign these systems into one of the groups and discuss which other aspects they support.[2] EdN:2

While most systems use one of the four aspects as the primary one, there are also some systems that focus primarily or even exclusively on the ontology. Therefore, use a fifth group for those systems.

2 Primary Aspect: Organization

[3] EdN:3

Every mathematical system employs a method for organizing a body of mathematical knowledge (MK). Organizational methods can be characterized by the following traits:

1. *MK units*, the kinds of entities that are used to organize the body of MK. Examples include algorithms, equations, tables, definitions, theorems, proofs, theories, and questions with answers.

2. *Foundation*, the underlying syntactic support (vocabulary and notation) and semantic support (concepts and assumptions) for expressing and understanding MK units. Examples include traditional mathematics practice; formal logics like first-order logic, simple type theory, set theory, and dependent type theory; and flexiformal foundations that support a combination of formal and informal elements.

3. *Creation mechanisms*, the ways of creating the MK units. Examples include definition principles, operations that combine simpler MK units into more complex MK units, and operations that produce MK units that are concrete instances of abstract MK units.

4. *Relation mechanisms*, the ways of relating the MK units with each other. Examples include various ways that one MK unit can be a subunit of another and meaning-preserving mappings from one unit to another.

[1] EdNote: MK: Possible outline:

1. The mathematics process involves several integrated activities.
2. Contemporary mathematical software systems usually focus on just one of these activities. Moreover, these systems are not designed to work with each other and do not employ a common knowledge base.
3. The limitations of contemporary systems have been exposed by the tremendous growth in the production of mathematical knowledge and "big math" projects like the Classification of Finite Simple Groups.
4. We need a mathematical software system that is holistic in the sense that it is designed to support the entire mathematics process and to be integrated with existing systems.
5. The first step towards the goal of a holistic system is to survey the mathematical software systems that are available today, which is subject of this paper.

[2] EdNote: MK: I do not think we should do it that way, but mention them in both

[3] EdNote: responsible: Bill, Yasmine

5. *Content control*, the way the content of the body of MK is controlled. Examples include central manager control or distributed user control.

6. *User orientation*, the kind of users that are served by how the body of MK is organized. Examples include users that consume MK and users that produce MK.

The following are common organizational methods.

Encyclopedia. MK is organized as an encyclopedia, as in, for example, the Encyclopedia of Mathematics [], PlanetMath [], and Wolfram MathWorld []. The basic MK unit is a mathematical article. The foundation is traditional mathematics practice. There are usually no special creation mechanisms. The main relation mechanism is the use of hyperlinks between articles. Content control is manager controlled. User orientation is for consumers.

Wiki. MK is organized as a wiki, as in, for example, in the set of mathematical articles in Wikipedia []. This organizational method is very similar to that of an encyclopedia except that content is user controlled and user orientation is for both consumers and producers.

Data Base. MK is organized as a collection of similar mathematical objects such as functions (as in the Digital Library of Mathematical Functions (DLMF) []), integer sequences (as in the Online Encyclopedia of Integer Sequences (OEIS) []), and triangle centers (as in the Encyclopedia of Triangle Centers (ETC) []. The MK units are the mathematical objects in the collection. The foundation is traditional mathematics practice. There are usually no special creation mechanisms. The main relation mechanism is the creation of connections between related members of the collection. Content is manager controlled. User orientation is for consumers.

Question-and-Answer Forum. MK is organized as a collection of questions and answers as in, for example, MathOverflow []. The main MK unit is a question about mathematics and one or more answers to the question. The foundation is traditional mathematics practice. There are creation mechanisms for submitting and rating questions, answers, and comments. The main relation mechanism is the use of tags to categorize and collect questions. Content is both manager and user controlled. User orientation is for both consumers and producers.

Algorithmic Theory. MK is organized as a collection of related numeric and symbolic algorithms as in, for example, Maple [] and Mathematica []. The main MK unit is an algorithm. The foundation is usually traditional mathematics practice. The main creation mechanism is for defining algorithms. The main relation mechanism is one algorithm calling another algorithm. Content is manager controlled. User orientation is for consumers.

Theory Graph. MK is organized as a directed graph whose nodes are axiomatic *theories* and whose directed edges are meaning-preserving mappings called *theory morphisms*. Special cases of this method are (1) a single theory with no theory morphisms and (2) a collection of theories with no theory morphisms between them. This organizational method is employed in most proof assistants, specification systems, and logical frameworks. The MK units are theories, theory morphisms, and the various subunits of a theory. The foundation is a formal logic, a collection of formal logics, or a flexiformal foundation. Creation mechanisms include operations for creating theories, theory subunits, and theory morphisms. Relation mechanisms relate a theory to the other theories that extend it and to the instances of it via theory morphisms. User orientation is usually for both consumers and producers.

3 Primary Aspect: Inference

Various methods have been developed to represent and perform inferences. We structure our presentation by how each method relates to computation, the aspect most whose integration with inference has drawn the most attention. In general, the ubiquity of underspecified function symbols and quantified variables means that logical expressions usually do not normalize to unique values. At best, computations like $y := f(x)$ can be represented as open-ended conjectures where different options for y are produced, each together with a proof of the respective equality. Therefore, inference systems usually sacrifice computation or at least its efficiency.

Proof assistants sit at the extreme end of this spectrum. They employ strong logics and high-level declarations to provide a convenient way to formalize domain knowledge and reason about it. The reasoning is usually interactive in order to represent inferences that are too difficult to be fully automated. Most proof assistants integrate at least some of the other methods to overcome this weakness.

Further along the spectrum, *automated theorem provers* use simpler logics than interactive proof assistants. They are fully automatic and much faster, but can handle much fewer theorems, and typically do not check their proofs. *Satisfiability checkers* continue this progression by aiming at decidable automation support, whereas theorem proving is usually an semi-decidable search problem. That limits them to propositional logic or specific theories of more expressive logics (usually of first-order logic) that are complete, i.e., where every formula can be proved or disproved. In the special cases, where satisfiability checkers are applicable, they come close to verified computation systems.

Orthogonal to the above triplet, there are several methods for realizing Turing-complete computation naturally inside a logic. Here imperative and object-oriented computation are usually avoided in favor of other programming paradigms that are easier to reason about. *Rewriting* aims at optimizing the $f(x) \rightsquigarrow y$ progression, allowing users to mark specific transformations as rewrite steps. *Terminating recursion* is the method of adding recursive functions to a logic in order to make it a pure functional programming language. Finally, *logic programming* restricts attention to theorems of a special form, for which proof search is simple and predictable so that users can represent computations by supplying axioms that guide the proof search.

In the sequel, we describe each method in some more detail.

Proof Assistants (C) The most successful proof assistants represent tens or hundreds of person-year investments into

- Define the foundational logic. Usually logics much stronger than textbook first- or higher-order logic are needed in practice. A big question has been the trade-off between flexible, untyped languages that rely on undecidable reasoning and rich type system that are more restrictive but have better computational properties. Typical choices are first-order set theory (e.g., Mizar [Miz]), higher-order logic (e.g., HOL [HOL], HOL Light [Har96], Isabelle/HOL [Pau94], PVS [ORS92]), and constructive type theory (e.g., Coq [Coq15], Matita [Asp+06], Lean [de +15], Agda [Nor05]). Some systems use undecidable type systems (e.g., Mizar, PVS) as compromises.
- Implement the logic. Usually the implementation starts with a kernel that checks proofs and then grows outwards in layers until a human-friendly surface syntax is exposed. Much work has been put into automatically filling in as many gaps left by the user as possible. Tactic languages (e.g., HOL, HOL Light, Coq, Isabelle) and high-level proof languages (e.g., Isabelle, Coq) and the integration of automated provers (e.g., Isabelle, Coq) and decision procedures (e.g., PVS, Isabelle) have been crucial here.
- Build a library of data structures. Usually proof assistants are only valuable if their standard library provides many data basic structures of mathematics (e.g., groups, real function) and computer science (e.g., records, inductive types). This wa most prominently envisioned in the QED manifesto [Qed]. But representing these and proving their characteristic properties has proved very expensive and remains a tough benchmark for the design of logic and system.

The tactic languages are often Turing-complete themselves (e.g., HOL Light, Isabelle, Coq). This is realized by writing tactics in the underlying programming language. Recently systems have tried to represent tactics in the systems itself either declaratively (e.g., Coq, Matita) or programmatically through reflection of the kernel data structures (e.g., Lean).

Most major proof assistants allow interspersed narrative structure, at the very least through comments. Some systems mimic sectioning where the scope of variables is determined not by the logic but by the narrative structure (e.g., Coq). Some systems (e.g., Isabelle, Agda) are narratively strong enough to make it feasible to write narrative documents (most importantly the documentation of the system itself) in the system.

[4] EdN:4

Automated Theorem Provers (A) Automated provers have been mostly developed for relatively simpler logics, where full automation is feasible. Therefore, they are often used as backend system integrated into, e.g., interactive proof assistants.

Most systems work with variants of first-order logic and compete regularly in the CASC competition. Examples are Vampire [RV02], E [Sch01], and Spass [Wei+02]. An ongoing trend is the gradual extension of first-order logic with additional features such as definitions, primitive numbers, types, and polymorphism.

Automated provers for higher-order logics are much harder to develop but are gaining strength. Example systems are Leo [Ben+08] and Satallax [Bro12].

Recently efforts have been made for automated provers to return checkable proofs in order to use them in proof assistants (see [Bla+16] for an overview). A big problem here is the selection of useful axioms to reduce the search space for the automated prover. Recently machine learning has been employed successfully for this purpose (e.g., [KU15]).

Satisfiability Checkers (S) While satisfiability is decidable for propositional logic and a few first-order theories (most importantly variants of linear arithmetic), the overall scope is limited. Decision problems are typically stated in the form of satisfiability problems and called *SAT solvers* (for propositional logic) or *SMT solvers* (satisfiability modulo theory, for specific theories of, typically, first-order logic). Specific decision procedure-based systems for higher logics include Z3 [dB08] and CVC [**cvc**]. [5] EdN:5

Satisfiability checkers for propositional logic have become so powerful that it is handle feasible to encode high-level problems as propositional problems by using large amounts of propositional variables.

To increase scope, efforts are made at combining decision procedures or applying them to theories with decidable fragments. While these efforts lose decidability, they may still be highly valuable in practice, e.g., by directly computing a value instead of extracting it from a proof, or to reduce the search space by eliminating unsatisfiable branches.

Terminating Recursions (T) Many proof assistants include sound and incomplete termination checkers (e.g., PVS, Coq, Agda) that only accept provably terminating functions. These are Turing-complete in the sense that the syntax can represent all computable functions but the termination checker will not accept all of them. Some systems (e.g., Isabelle, Coq) can export those programs in external programming languages after verifying their correctness in the logics.

In some cases it can be hard to draw the line between inference and computation systems. This is the case for systems that use functional programming combined with the representation of proofs-as-programs (e.g., Coq, Agda).

[4] EdNote: JC: recent paper "A Survey on Theorem Provers in Formal Methods" - https://arxiv.org/abs/1912.03028 has just appeared. We should at the very least cite all the software that appears in it.

[5] EdNote: FR: cite some SAT and SMT solvers here

Rewriting (R) Rewriting implements a directed equality relation between expressions. This can be used both to obtain Turing-complete computation and for reasoning (by rewriting formulas to a boolean value).

Compared to other forms of computation, rewriting is very inefficient, e.g., rewriting polynomials into normal form takes exponentially longer than an algorithm based on plain "arithmetic". But the embedding of rewrite systems in logic permits proving the soundness of each rewrite rule.

Originally most rewrite systems were based on first-order logic due to its decidable unification (e.g., Maude [Cla+96]).[6] But recently more systems are employing rewriting in higher-order setting (e.g., Dedukti [BCH12]). [7] EdN:6 EdN:7

A lot of effort has gone into establishing the confluence and termination of sets of rewrite rules. This is critical to establish that rewriting implements a deterministic computation. Therefore, many systems try to establish confluence and termination of sets of rewrite rules automatically. [8] EdN:8

Many proof assistants integrate rewrite engines that use some of the proved theorems as rewrite rules that are applied automatically by the system (e.g., PVS, Isabelle).

Logic Programming (L) If the proof search behavior of a prover is known to the user and predictable in practice, this can be instrumented for computation. This works particularly well with axioms in Horn-form where searching a proof of the conclusion triggers searching proofs of the assumptions. This is the basic idea of logic programming, where a formalization consists of a set of Horn axioms, which can be seen as both a specification and (via predicatble proof search) as a program. If special predicate symbols are added, whose "proofs" consists of extra-logical operations like I/O side effects, this yields a general purpose programming language.

Logic programming was mostly investigated in untyped first-order logic via various Prolog dialects [**prolog**]. But higher-order variants exists as well, e.g., such as λProlog [Mil]. [9] EdN:9

[10] EdN:10

4 Primary Aspect: Computation

[11] [12] EdN:11

Computation is a rather broad topic: for example, term rewriting systems and finite state machines are often regarded as performing computations. At the Turing-complete end of the spectrum, we could list all programming languages as performing computations — because, well, they do! EdN:12

As our primary focus is still centered on mathematics, this helpfully narrows things down. Even though one can indeed use just about any language to do mathematics, it makes sense to instead focus on those languages that have been designed with mathematics in mind.

The systems can be usefully divided according to the kinds of data they were primarily designed to handle:

- Typed
- Symbolic
- Algebraic
- Numeric
  - analytic

[6] EdNote: add more citations
[7] EdNote: check Cynthia Kop's talk on higher-order rewriting for more citations
[8] EdNote: cite examples
[9] EdNote: need more references here
[10] EdNote: Responsible: Florian
[11] EdNote: Responsible: Jacques
[12] EdNote: MK: We need letters for the big table here. I will use "T" for Turing Complete; need that for TEX

– statistical

By *typed* data, we mean data that can be expressed in "type theory", as originating from Russell and Whitehead [WR10], through Church [Chu40], until today; the historical development until 10 years ago is well documented in [KLN04]. Although we have seen a wide varieties of type theories used for systems that perform inference, for systems that take computations seriously, one family emerges: dependent type theories. By *symbolic* data, we mean data that can be best expressed as *abstract syntax trees*, usually containing "free variables". From a type-theoretical point of view, these are significantly harder to deal with, as these would then form *open terms*, which are notoriously difficult to manipulate correctly. This is why all know symbolic computation systems are untyped. By *algebraic*, we basically mean data that belongs to the mathematical subdomain of Algebra. What distinguishes these is that, although it is frequently convenient to use "free variables" for the visual display of these objects, they are not fundamentally required for an adequate representation. Under Algebra, we also include systems that do exact computations on natural numbers, integers, Gaussian integers, etc, as these are also algebraic. By *numeric*, we mean systems whose data include "real numbers" in one way or another; it is useful to subdivide this class further, into the systems that specialize in more analytic problems (quadrature, differential equations) from those that deal with statistics. Both kinds excell at computations based on linear algebra.

Some of the systems that we survey below are quite broad, and so implement many features in common: one can indeed do statistics in Maple, and symbolic computation in Matlab. The classification is not meant to "squeeze" any system into a narrow box, but rather to express the *fundamental organizational system* around which the system grew outward to encompass much more.

Typed Agda [Nor05], Idris [Bra13]
ATS.

Symbolic Mathematica
Maple
Axiom

Exact Computation Systems Gap: groups
Singular: ideals
Sage: integrated with Python and (via Python) other languages

Scientific Computation Matlab, scilab, Octave.
ChebFun, NumPY,

Statistics Packages R. SPSS, SAS, Minitab.

More categories Machine learning, probabilistic programming. Term rewriting?

commentary [13] By *reflection*, we mean the ability of a logic to have access to (some of) the reasoning facilities usually associated with its meta-logic. Computational reflection is similar, in the context of programming languages; this usually proceeds via a representation of some (otherwise opaque) concepts, which can be manipulated, before being *reified*. If the representation is adequate and the manipulations are meaning preserving, then computation can implement reasoning. EdN:13

If programming languages use a sufficiently strong *type system*, they may be able to embed deduction into computation. Types are, via Curry-Howard, (simple) propositions that are considered

[13] EdNote: while I understand why this was put here, I think this should just be deleted. In many ways, this is anti-tetrapodian thinking, as it priviledges inference over everything else.

true if they are inhabited. Then type checking is a form of verification (i.e., checking that a particular proof/term inhabits a particular proposition/type). *Abstract interpretation* moves beyond "simple" types into being able to attach significantly more powerful properties to programs.

5 Primary Aspect: Concretization

[14]

EdN:14
BNP:15

Overview The naming of this aspect of knowledge section has proved surprisingly difficult. We mean to include any practice of representing mathematical objects in terms of concrete data structures. Mathematics has a long tradition of such efforts, going back to, e.g., clay tablets of Pythagorean triples, lists of decimal digits of π, or logarithm tables. More modern incarnations include computer-supported practices like large prime numbers, the database of integer sequences, and the enumeration of isomorphism classes of simple finite groups. But this practice does not a standard name.

By *concrete data structures*, we mean any data formed using primitive objects such as integers and strings as well as complex constructors like lists, records, and tables. These objects have in common that they have an objective physical reality that is beyond doubt: for example, any particular finite list of integers exists absolutely, whereas the existence of, e.g., proofs or programs may be relative to philosophical assumptions (e.g., impredicativity, classical reasoning, axiom of choice) or mathematical conditions (e.g., soundness of an argument, termination of an algorithm). Thus, our concrete objects are tangible and material in a way that is opposite to the Platonic objects that deduction and narration and to some extent computation are concerned with. Thus, we can also think of them as the shadows of Platonic philosophy or as Aristotelian objects in the sense of being empirical, observable, and practical.

Most of the time, concrete objects are aggregated in the form of tables such as logarithm tables or the many large relational databases described below. We considered *tabulation* as an alternative name for this aspect but opted against it to avoid excluding other representation languages for concrete data such as arrays and JSON.

Encodings There are two ways to define the semantics of concrete objects. Firstly, we can characterize the concrete objects as the subset of those mathematical objects that are self-denoting: e.g., any natural number is interpreted as itself. This is in contrast to the other three aspects, where interpretations are needed to map objects to their denotation. Secondly, many non-concrete mathematical objects can be represented in terms of concrete ones, a process that we call *encoding*. Encodings are commonly used in databases of algebraic structures such as elliptic curves or isomorphism classes of graphs. More generally, any effective representation of mathematical objects requires some encoding as concrete objects because only those can be acted on by computers. Therefore, in general, the semantics of concrete objects consists in applying the dual *decoding* operation. We speak of *codecs* for a pair of encoding or decoding that represent a class of mathematical objects as concrete ones.

Data and Knowledge [16] We use the following general terms for all aspects: *Syntax* is a set of rules for forming objects, and *data* is any piece of well-formed syntax. *Semantics* is a set of rules for interpreting objects, and *knowledge* is a pair of a datum and its semantics.

EdN:16
ENP:15

Until the advent of computer-supported mathematics, the creation and sharing of data only received attention in passing. The logarithm tables are a good example of this, as are the examples collected in the community effort initiated by Gordon Royle on MathOverflow. (https://mathoverflow.net/questions/47044/what-are-some-early-examples-of-creation-of-lists-catalogues-of-pa

[14] EdNote: responsible: use summary of Katja's survey
[15] New Part: FR: the summary of our discussion
[16] EdNote: FR@MK: This should go to the introduction, just putting it here for now to keep the changes together

Billey and Tenner introduced the concept of a fingerprint database of theorems in 2013 [BT13]. The primary example of this is the OEIS. They stress the importance of the following aspects: searchability, collaborativeness, citability of the contents, and indexing by small, language independent and canonical data.

Mathematical datasets and databases today are highly varied and range from small to large in several aspects. The datasets can easily reach the Gigabyte range: LMFDB (1TB data in number theory), or a lattice dataset by Kohonen (uncompressed about 1.5TB of lattices), to name some of the largest. The can range from short lists of objects that are extremely hard to compute, to gigantic lists of millions of objects, such as the GAP Small Groups Library (about 450 million finite groups) and the previously mentioned dataset of $17 \cdot 10^9$ lattices. Similarly, the authorship varies from single-author datasets, to community efforts such as the OEIS, with thousands of contributors. The structure of mathematical datasets can be as simple as having a list of objects, a list of records that can contain information like mathematical invariants in addition to the object, to complex databases of related tables such as the LMFDB.

Commonly occurring themes in mathematical data are ad-hoc implementations of codecs, lack of community guidelines for data, and similar.[17] EdN:17

Mathematical data can be roughly divided into three main categories of tabular, linked and symbolic data.

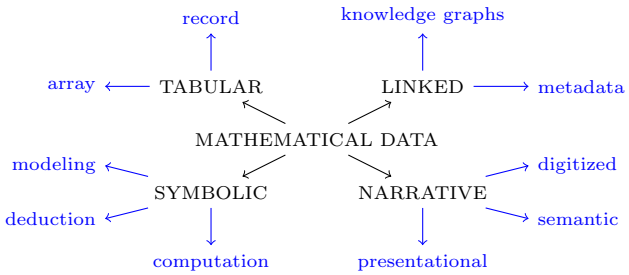

Figure 2: Kinds of mathematical data

Tabular data (T) These employ representation theorems that allow encoding mathematical objects as simple data structures built from numbers, strings, lists, and records. Thus, contrary to the other two kinds of mathematical data, tabular data combines optimized storage and processing with capturing the whole semantics of the objects. But such representation theorems do not always exist because sets and functions, which are the foundation of most mathematics, are inherently hard to represent concretely. Moreover, representation theorems may be very difficult to establish and understand, and there may be multiple different representations for the same object.

Tabular data can be subdivided into record data, where datasets are sets of records conforming to the same schema, and array data, which consists of very large, multidimensional arrays that require optimized management. Array data tends to come up in settings with large but simply-structured datasets such as simulation time series, while record data is often needed to represent complex objects, especially those from pure mathematics. Record data and querying is very well-standardized by the relational (SQL) model. However, if encodings are used, SQL can never answer queries about the semantics of the original object. The younger array data bases, which offer efficient access to contiguous — possibly lower-dimensional — sub-arrays of datasets (voxels), are less standardized, but OPenNDAP is becoming increasingly recognized even outside the GeoData community, where it originated.

Mathematical databases like the L-functions and Modular Forms DataBase (LMFDB), the GAP Small Groups Library, or the Open Encyclopedia of Integer Sequences (OEIS) store millions of mathematical objects together with their semantic properties, both human-curated or machine-generated.

[17] EdNote: Mention FAIRMat?

Linked data (L) These introduce identifiers for objects and then treats them as blackboxes, only representing the identifier and not the original object. The internal structure and the semantics of the object remain unspecified except for maintaining a set of named relations and attributions for these identifiers. The named relations allow forming large networks of objects, and the attributions of concrete values provide limited information about each one. Linked data can be subdivided into knowledge graphs and metadata, e.g., as used in publication indexing services.

As linked data forms the backbone of the Semantic Web, linked data formats are very well-standardized: data formats come as RDF, the relations and attributes are expressed as ontologies in OWL2, and RDF-based databases (also called triple-stores) can be queried via SPARQL. For example, services like DBPedia and Yago crawl various aspects of Wikipedia to extract linked data collections and provide SPARQL endpoints. The WikiData database collects such linked data and uses them to answer queries about the objects.

Mathematical information services like zbMATH, EuDML, and swMATH extend bibliographic metadata of mathematical publications with math subject classifications (essentially taxonomic semantic information) and use automatic extraction to give users enhanced, semantic search capabilities. Also notable is Wikidata, with 34 GB linked data, thereof about 4K formula entities, interlinked, e.g., with named theorems, persons, and/or publications.

Others: arXiv, RADAR, (GitHub, Wikimedia?)
[18]

EdN:18

Symbolic data (S) These consist of formal expressions such as formulas, formal proofs, programs, graphs, diagrams, etc. These are written in a variety of highly-structured formal languages specifically designed for individual domains. Because it allows for abstraction principles such as underspecification, quantification, and variable binding, symbolic data can (in contrast to the other two kinds) capture the full semantics of mathematical objects. This comes at the price of being context-sensitive: expressions cannot be easily moved across environments.

Working with symbolic data in mathematics can be subdivided based on the area of application into modeling, deduction, and computation. Each area employs a wide variety of sophisticated formal languages: modeling languages, logics, resp. programming languages. Multiple different, often mutually non-interoperable, representation formats have been developed for symbolic data, usually growing out of small research projects and reaching different degrees of standardization, tool support, and user following. These are usually optimized for specific applications, and little cross-format sharing is possible. In response to this problematic situation, standard formats have been designed such as MathML and OMDoc/MMT.

Libraries of formalized mathematics directly specify the meaning of mathematical definitions, theorems, and proofs in a machine-verifiable way. Tens of thousands of such formal proofs are available in libraries such as OAF, Modelica [19].

EdN:19

BOP:20

Narrative Data (N) consists of mathematical documents and text fragments. We speak of **mathematical vernacular** for the peculiar mixture of mathematical formulae, natural language with special idioms, and diagrams. There are four levels of formality of narrative data:

0. **written up**: for communication or archival purposes,
1. **digitized:** scanned into images from documents,
2. **presentational:** represented in a form that allows flexible presentation on electronic media, such as web browsers; born digital or OCRed from digitized ones,
3. **semantic**: in a form that makes explicit the functional structure and the relations between formulae, the objects they denote and the mathematical context.

All levels of formality are relevant for mathematical communication, but machine support for reasoning and knowledge management can only be given at the semantic level. We could extend

[18] EdNote: Any other datasets/services?
[19] EdNote: others?
[20] Old Part: MK: totally rework

this classification by a fourth level of narrative data for formalized documents; but these abstract from the narrative form and are therefore counted as symbolic data.

Note that we can always go from higher levels to lower ones, by styling: presenting semantic features by narrative patterns. Therefore we also count such patterns as narrative data – e.g. **notation definitions** such as $\binom{n}{k}$ or $\mathcal{C}_k^n$ for the binomial coefficients or verbalizations in different languages. EOP:20

Standalone Databases OEIS [OEIS]: some ontology and narration

LMFDB [LM]: content produced by computation; filled by computation; computation integrated into frontend via Sage; narration and ontology for background knowledge

FindStat [BSa14]:

ATLAS of Finite Group Representations (http://for.mat.bham.ac.uk/atlas/):

Database of Ring Theory (http://ringtheory.herokuapp.com/):

Math Counterexamples (http://www.mathcounterexamples.net/):

Manifold Atlas: (http://www.map.mpim-bonn.mpg.de)

Distributome (http://www.distributome.org/)

SuiteSparse (https://sparse.tamu.edu/)

House of Graphs (http://hog.grinvin.org)

Digital Library of Mathematical Functions (https://dlmf.nist.gov/)

Data Sets within Computation Systems Mathematics (embedded and external data sources)

Sage

GAP data libraries (http://www.gap-system.org/Datalib/datalib.html): (e.g., table of transitive groups) BOP:21

Figure 3 gives an overview over the scale of the problem and an overview over some state-of-the-art datasets. EOP:21

6 Primary Aspect: Narration

[22] The "narration" aspect of the tetrapod is concerned with mathematical knowledge in a form that can be digested by humans. Narratively represented mathematical knowledge usually exists in mathematical documents[1] like articles, books, or preprints, that expose the ideas in natural language but also in diagrams, tables, and simulations. While rigour and correctness are important concerns in narration, the main emphasis is on communicating ideas, insights, intuitions, and inherent connections efficiently to colleagues well-versed in the particular topic or students who want to become that. As a consequence, more than half of the text of a typical mathematical document consists of introductions, motivations, recaps, remarks, outlooks, conclusions, and references. Even though the "packaging" of mathematical knowledge into documents leads to some duplication in the mathematical literature, it seems to be an efficient way of dealing with communication and knowledge preservation and can thus be seen as a necessary overhead in scholarly communication. EdN:22

The primary aim of this survey is to explore holistic mathematical software systems – here software support for mathematical documents. This in turn depends on the depth of explicit structural and semantic markup in documents. Currently, there are four (plus one) levels of representation of mathematical documents:

RL0. **written up**: for communication (e.g. chalk on blackboards) or archival purposes – e.g. on papyrus scrolls $\widehat{=}$ ∼90%[2] of the mathematical documents.

[21] Old Part: integrate the figure; we want that, not quite in this way

[22] EdNote: responsible: Michael

[1] While we take a rather inclusive view on "documents", we limit ourselves to written documents, excluding audio recordings and videos. Our rationale is that this does not constitute a loss of generality since the latter could be transcribed into written documents without significant loss in meaning.

[2] The percentages in this classification are rough estimates that should be taken as qualitative indications of the relative size rather than actual quantities.

Dataset	Description
Symbolic Data	
Theorem prover libraries	$\approx$ 5 proof libraries, $\approx 10^5$ theorems each, $\approx$ 200 GB
Computer algebra systems	e.g., SageMath distribution bundles $\approx$ 4 GB of various tools and libraries
Modelica libraries	$>$ 10 official, $>$ 100 open-source, $\approx$ 50 commercial, $>$ 5.000 classes in the Standard Library, industrial models can reach .5M equations
Tabular Data	
Integer Sequences	$\approx$ 330K sequences, $\approx$ 1 TB
Sequence Identities	$\approx$ .3M sequence identities, $\approx$ 2.5 TB
Highly symmetric graphs, maps, polytopes	$\approx$ 30 datasets, $\approx 2 \cdot 10^6$ objects, $\approx$ 1 TB
Finite lattices	7 datasets, $\approx 17 \cdot 10^9$ objects, $\approx$ 1.5 TB
Combinatorial statistics and maps	$\approx$ 1.500 objects
SageMath databases	12 datasets
L-functions and modular forms	$\approx$ 80 datasets, $\approx 10^9$ objects, $\approx$ 1 TB
Linked Data	
zbMATH	$\approx$ 4M publication records with semantic data, $\approx$ 30M reference data, $>$ 1M disambig. authors, $\approx$ 2, 7M full text links: $\approx$ 1M OA
swMATH	$\approx$ 25K software records with $>$ 300K links to $>$ 180K publications
EuDML	$\approx$ 260K open full-text publications
Wikidata	34 GB linked data, thereof about 4K formula entities, interlinked, e.g., with named theorems, persons, and/or publications
Narrative Data	
arXiv.org	$\approx$ 300K math preprints (of $\approx$ 1.6M) most with LaTeX sources
EuDML	$\approx$ 260K open full-text publications, digitized journal back issues
MathOverFlow	$\approx$ 1, 1M questions/answers, $\geq$ 11K answer authors
Stacks project	$\geq$ 6000 pages, semantically annotated, curated, searchable textbook
nLab	$\geq$ 13K pages on category theory and applications

Figure 3: Summary of mathematical datasets

RL1. **digital** usually digitized from print – e.g. as TIFFs $\widehat{=}$ ∼50%
RL2. **presentational**: encoded text interspersed with presentation markup – e.g. PDF, Word, TEX or presentation MathML $\widehat{=}$ ∼20%
RL3. **semantic**: encoded text with functional markup for the meaning, e.g. LATEX, STEX, Mathematica Notebooks $\widehat{=}$ ∼1%
RL4. **formal**: The meaning of the document is fully specified and thus machine-actionable at all levels. ≤0.1%

We remark that the delineation of levels is somewhat fuzzy and that the "levels" themselves are far from uniform. Nevertheless they constitute a useful categorization for our discussion. Especially in the higher levels, the presentational and semantic markup is usually restricted to particular aspects of the document functionality. We have ordered the examples of representation formats by increasing opportunities for structural and functional markup.

Note transforming from higher levels to lower levels is usually simple via largely context-free **styling** rules, whereas the opposite direction – **semantics extraction** – involves non-trivial context-dependent heuristic choices. For instance in the transformation between the levels RL1. and RL2. we have the difference between printing (down transformation) or OCR (optical character recognition; up transformation).

We finally note that computation and thus machine support needs explicitly represented structures and thus higher representation levels lead to more opportunities for software support. Therefore we will structure our survey bottom-up in the hierarchy above, starting with level RL2. since levels RL0. and RL1. have no discernable math-specific aspects.

At the Presentational Level (P): Word Processors & Document Preparation Systems
At this level, we have any kind of software system and for document preparation that can deal with **mathematical vernacular**, the peculiar mixture of natural language, mathematical formulae, and diagrams digitally. In contrast to the image-based formats at level RL1., text is encoded as sequences of characters, whereas formulae and diagrams are in some kind of presentational markup. This is sufficient to e.g. make documents at the presentation level searchable in conventional bag-of-words-based search engines like Google or Bing. The systems and representation formats mainly differ in their

1. **authoring model**: WYSIWYG or formatted/programmable text,
2. **target media**: paginated or flexible page size,
3. treatment of mathematical formulae.

Word processors like MS Word or LibreOffice Writer implement a WYSIWYG – "what you see is what you get" – authoring model and target paginated media. Typesetting systems like TEX/LATEX, the document preparation system preferred in mathematics, let authors encode documents as Unicode strings with executable – often user-definable – control sequences, which a formatter expands into a primitive page description format. For publication, the standard target page description format in both cases is usually PDF (Adobe's standardized Portable Document Format). This fixes page layouts down to the character position. Diagrams and formula components that do not come from one of the available fonts are represented as vector graphics. In particular, mathematical formulae lose all structural information during the PDF transformation, so that higher-level services like mathematical search or screen readers have nothing to go on, and would OCR-like facilities to function. In essence, mathematical formulae and diagrams are still at level RL1 (image-like) in PDF, even though the "source" (Office Open XML [OOXML06] for MS Word and Open Document Format [DB12] for LibreOffice Writer) may still have had the necessary structures.

Most document preparation systems also allow the export of HTML5 [Fau+17], a web markup format for interactive multimedia documents, that can encode mathematical formulae via MathML (the Mathematical Markup Language; see [MML310]) and diagrams via SVG (Scalable Vector Graphics; see [Dah+11]). For MS Word, MathML export is a native feature, for LibreOffice Writer via a plugin Writer2xhtml [W2X], TEX/LATEX can be exported to HTML5 (and the eBook format ePUB3 [Con+11] based on it) via the LATEXML engine or TEX4HT. Note that MathML has two

sub-languages: **presentation MathML** specifies the visual layout of formulae (i.e. level RL2), and **content MathML** for the meaning (the associated operator trees; i.e. RL3). The exports above all restrict themselves to presentation MathML (though LaTeXML [LTX] does a best-effort attempt at inferring content MathML). But even presentation MathML has enough structure to support formula screen readers like MathPlayer [Mat] or mathematical search engines; see [GSC15; Aiz+16] for pointers to the state of the art and [ZBF] for an online example.

Formal, Narrative Documents (F) To understand the semantic representation level (RL3) of narrative mathematical documents, which mixes language and with formal aspects, it is good to think about fully formal narrative documents. By definition, these could consist of a sequence of logical propositions[3], possibly extended by a formalization of sectioning, discourse, and rhetorical structures. To the best of the knowledge of the authors, there are no fully formal, narrative representations of matheamtical knowledge. All fully formal representations (e.g. proof assistant libraries cf. Section 3) are organized with respect to the inherent structures of the knowledge space, with little concern to a given narrative for human readers. Where attempts of a narrative are made – e.g. in comments – these are informal natural language, not logic. A notable example is van Benthem Jutting' s formalization [BJ77] of Landau's "*Grundlagen der Analysis* [Lan30], where the formalization closely follows the structure of the original. Arguably the narrative structure of the narrative structure of the Grundlagen is very limited, and the formalization excluded the explanatory introduction. The NaProChe [Cra+10; CKS11] project develops a controlled natural language for mathematics[23], i.e. a formal language that is – syntactically – a subset of mathematical vernacular, with the aim of verifying mathematics in the Isabelle proof assistant. Again, introductions, motivations, recaps, remarks, outlooks, conclusions, and references are not part of the language. EdN:23

A similar but dual example – verbalization instead of formalization – is the case of Mizar articles, from which human-oriented presentations for the Journal of Formalized Mathematics [JFM] are generated. Again, abstracts and introductions have to supplied by (human) authors. The remaining content is generated from the Mizar theorems and proofs. The latter are sequences of statements and justifications which can be verified by the Mizar prover. In fact, formal proofs given as proof step sequences may be the only fully formal narrative documents currently available. In contrast to proof objects – λ terms in an expressive type theory – they combine full formality with a narrative (step-by-step with justifications) structure conducive to human understanding. Generally, proof presentation – i.e. transforming proofs to mathematical vernacular – has been studied in various contexts; see [ABR01; Hua96; Hor00] for details and pointers.

A particularly influential design has been the ISAR[4] (Intelligible semi-automated reasoning) proof document language [Wen07] of the Isabelle proof assistant. ISAR tries to bridge the semantic gap between prover-internal – proof objects generated by tactic scripts – and an appropriate level of abstraction for user-level work. ISAR proof texts consist document constructors, atopic steps via high-level tactic invocations, and library references. Thus they admit a purely static reading, thus being intelligible later without requiring dynamic replay that is so typical for traditional proof scripts. This is a very important characteristic of narrative representations.

Semi-Formal Systems (S): Documents at the Semantic Level The semantic level relaxes the requirement of full formality and allows informal elements interspersed with formal ones. For instance, Isabelle provides native syntax for LaTeX-like commands as well as raw LaTeX, and all Isabelle documents can be turned into LaTeX for documentation. Similarly, Agda [Nor05] can read two kinds of files: documentation files with interspersed Agda code or Agda code files with interspersed documentation.

Thus semiformal systems can choose which aspects to formalize and focus on services using those, leaving the informal ones to humans; [Koh13] introduces the concept of **flexiformality**

[3] In fact, the resolution of the Grundlagen Crisis of Mathematics in the last century is that any mathematical document can – in principle – be formalized in first-order logic with axiomatic set theory.

[23] EdNote: MK: does it have a name?

[4] inspired by the Mizar language, hence the name

(flexible formality) and discusses the issues involved. A good example is weak type theory [KN04], a λ-calculus with a linguistically inspired type system, which is intended as an intermediate step in the formalization of mathematical developments. The MathLang system [Kam+14] based on ideas from weak type theory allows to annotate (i.e. flexiformalize) various aspects of a mathematical document, up to a point where enough semantic information to drive verification of the document in a proof assistant [Ret09].

The OMDoc (Open Mathematical Documents) format [Koh06] is possibly the most complete framework for flexiformal mathematics. It subsumes all the other representation formats and can serve as an interoperability layer. It specifies markup for mathematical documents and knowledge in a system-independent general framework and relates the two aspects. OMDoc provides representational infrastructure at three levels: the object level for mathematical formulae, based on OpenMath[24] and content MathML, the statement level for definitions, theorems, and proofs, and the theory level for document sectioning and knowledge grouping. OMDoc documents can be created by writing them in STEX [Koh08; sTeX], a variant of LATEX that allows "semantic preloading", i.e. invisible OMDoc markup in the LATEX sources and then converting to OMDoc/XML via the LATEXML system. Conversely, OMDoc documents can be transformed into **active documents**, which use the semantic information for embedded services – the more semantic preloading, the more services – see [Koh+11] for a discussion. EdN:24

Finally, another example of flexiformal – here computational – documents are Wolfram Notebooks [MNB] or Jupyter [JN] notebooks. Here, computational "cells" with executable code (Mathematica for Wolfram notebooks and a wide variety of computational systems for Jupyter notebooks) are interleaved with text cells, which provide an (informal) narrative. Computational cells show the results of computations, the code can be arbitrarily edited and re-executed, giving a very flexible way of exploring the mathematical contents. Computational cells can also drive "widgets", which pipe computation results into special-purpose interaction forms.

7 The Big Table of Systems and their Aspects

[25] EdN:25

[26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] EdN:26

We give a overview of mathematical software systems from a tetrapod perspective in Table 1. We list systems in the first row, and specify which aspects they support in the last five using the letter codes specified in the "sub-aspects" in Sections 2 to 6. Note that the letter codes are only unique per column.[38] Where necessary, we mark the codes with comments which can be referenced in the list below. EdN:27 EdN:28 EdN:29 EdN:30 EdN:31

1 TEX [Knu84] pairs a set of layout primitives with a Turing-complete macro expansion facility, which is use d by a large community to define libraries of macros. LATEX [Lam94] is the widely used one; it establishes semantic markup for sectioning, crossreferences, bibliographic references, and statements. EdN:32 EdN:33 EdN:34 EdN:35

EdN:36 EdN:37 EdN:38

[24] EdNote: MK: introduce OpenMath somewhere, probably in the inference aspect
[25] EdNote: Conjecturing, Theory Exploration
[26] EdNote: not Python
[27] EdNote: Matlab, Wolfram Alpha? Geogebra?
[28] EdNote: Geogebra represents Graphing Calc
[29] EdNote: Numeric/Scientific computation, Optimisation, Statistics: refer to surveys?
[30] EdNote: Keep in mind: IDRIS, F*
[31] EdNote: Otter representative of Prover 9, Mace
[32] EdNote: probabilistic proof? is it a computation thing?
[33] EdNote: zbMath representative for MathSciNet, has swMath, Google Scholar, Scopus, arXiv
[34] EdNote: MathTutor History of Mathematics (bibliographies) archive http://www-history.mcs.st-and.ac.uk/
[35] EdNote: not Oracle
[36] EdNote: not InDesign, Markdown
[37] EdNote: MathWorld
[38] EdNote: Make an example where this happens

System	Reference	Organization	Reasoning	Computation	Tabulation	Narration
MathOverflow		F				
Polymath		F				
AFP		K				
MSC		S				
MathHub						
Coq	[Coq15]		C A T P R			
Isabelle			P			S
Mizar			P			
Otter			A			
* Hammer			A			
CVC/Z3						
Prolog			L			
FOIL			L			
Mathematica						S
SageMath						
GAP						
GeoGebra						
CoCalc						
Octave						
Simulink						
R						
Stan						
zbMath		S N			L	P
Math Genealogy Project					L	
WikiData		K			L	
OEIS		E			T	P
LMFDB					T	
Small Groups Library					T	
arXiv					N	P
Mizar Library					S	
DLMF					S	S
Inverse Symbolic Calculator					S	
TEX/LATEX	[Knu84; Lam94]			[1]		P S[1]
pMathML	[MML310]					P
STEX	[Koh08]					P S F
SIUnitsX	[Wri]					F
OpenMath/cMathML	[Bus+04; MML310]					F[2]
Wikipedia	[Wik]	E[3]				PS[4]

Table 1: Tetrapod Systems and their Aspects

2 OpenMath and – by reference – presentation MathML (whose semantics is given in terms fo OpenMath) fully describe the structure of mathematical formulae and give the meaning of the symbols in terms of OpenMath Content Dictionaries, which are semantic mathematical documents themselves.
3 Wikipedia uses a restricted subset of TEXto create presentation MathML.
4 Some of the links and concepts are classified with Linked Open Data annotations.

8 Realizing Secondary Aspects

Systems usually use additional aspects, which we call secondary. The secondary aspect can have multiple roles:

- It might enhance knowledge written in the primary aspects, e.g., narrative documentation of programs.
- It may substitute for knowledge written in the primary aspects, e.g., a narrative snippet describing an omitted proof step in a semi-formal proof.
- It may be used to talk about knowledge written in the primary aspect, e.g., a computational tactic that produces proofs or the verification of a computational system.

Primary Aspect	Secondary Aspect				
	Org	Inf	Comp	Tab	Narr
Org	category of theories module systems	type inference			
Inf	theories	meta-theorems verification proof-checking	recursion rewriting logic programming tactics ATP decision procedures		documentation semi-formal proofs
Comp	specifications	verification	preprocessing code generation profiling	memoization package repositories	documentation
Tab	schemas		querying built-in functions		
Narr			active documents literate programming macros		documentation

Figure 4: Paradigms for Supporting Secondary Aspects

Note that some of these roles allow for the primary and secondary aspect to be the same. For example, we can use a computational preprocessor to generate programs before compilation. Or we can proof a meta-theorem that states the admissibility of an additional inference rule.

9 Conclusion

[39]

EdN:39

References

[ABR01] Ahmed Amerkad, Yves Bertot, and Laurence Rideau. "Mathematics and Proof Presentation in Pcoq". In: *Proceedings of the Workshop on Proof Transformation, Proof Presentations and Complexity of Proofs (PTP-01)*. Ed. by Uwe Egly et al. Universitá degli studi di Siena, 2001, pp. 51–60.

[Aiz+16] Akiko Aizawa et al. "NTCIR-12 MathIR Task Overview". In: *Proceedings of the 12th NTCIR Conference on Evaluation of Information Access Technologies*. Ed. by Noriko Kando, Tetsuya Sakai, and Mark Sanderson. Tokyo, Japan: NII, Tokyo, 2016, pp. 299–308. URL: http://research.nii.ac.jp/ntcir/workshop/OnlineProceedings12/pdf/ntcir/OVERVIEW/01-NTCIR12-OV-MathIR-ZanibbiR.pdf.

[Asp+06] A. Asperti et al. "Crafting a Proof Assistant". In: *TYPES*. Ed. by T. Altenkirch and C. McBride. Springer, 2006, pp. 18–32.

[BCH12] M. Boespflug, Q. Carbonneaux, and O. Hermant. "The λΠ-calculus modulo as a universal proof language". In: *Proceedings of PxTP2012: Proof Exchange for Theorem Proving*. Ed. by D. Pichardie and T. Weber. 2012, pp. 28–43.

[39] EdNote: MK: to be written

[Ben+08] C. Benzmüller et al. "LEO-II - A Cooperative Automatic Theorem Prover for Classical Higher-Order Logic (System Description)". In: *Automated Reasoning*. Ed. by A. Armando, P. Baumgartner, and G. Dowek. Springer, 2008, pp. 162–170.

[BJ77] L.S. van Benthem Jutting. "Checking Landau's "Grundlagen" in the AUTOMATH System". PhD thesis. Eindhoven University of Technology, 1977.

[Bla+16] J. Blanchette et al. "Hammering towards QED". In: *Journal of Formalized Reasoning* 9.1 (2016), pp. 101–148.

[Bra13] E. Brady. "Idris, a general-purpose dependently typed programming language: Design and implementation". In: *Journal of Functional Programming* 23.5 (2013), pp. 552–593.

[Bro12] C. Brown. "Satallax: An Automatic Higher-Order Prover". In: *Automated Reasoning*. Ed. by B. Gramlich, D. Miller, and U. Sattler. Springer, 2012, pp. 111–117.

[BSa14] C. Berg, C. Stump, and al. *FindStat: The Combinatorial Statistic Finder*. http://www.FindStat.org. [Online; accessed 31 August 2016]. 2014.

[BT13] Sara C. Billey and Bridget E. Tenner. "Fingerprint databases for theorems". In: *Notices Amer. Math. Soc.* 60.8 (2013), pp. 1034–1039. ISSN: 0002-9920. DOI: 10.1090/noti1029.

[Bus+04] Stephen Buswell et al. *The Open Math Standard, Version 2.0*. Tech. rep. The OpenMath Society, 2004. URL: http://www.openmath.org/standard/om20.

[Car+19] Jacques Carette et al. "Big Math and the One-Brain Barrier – A Position Paper and Architecture Proposal". In: *Mathematical Intelligencer* (2019). in press. URL: https://arxiv.org/abs/1904.10405.

[Chu40] Alonzo Church. "A Formulation of the Simple Theory of Types". In: *Journal of Symbolic Logic* 5 (1940), pp. 56–68.

[CKS11] Marcos Cramer, Peter Koepke, and Bernhard Schröder. "Parsing and Disambiguation of Symbolic Mathematics in the Naproche System". In: *Intelligent Computer Mathematics*. Ed. by James Davenport et al. LNAI 6824. Springer Verlag, 2011, pp. 180–195. ISBN: 978-3-642-22672-4.

[Cla+96] M. Clavel et al. "Principles of Maude". In: *Proceedings of the First International Workshop on Rewriting Logic*. Ed. by J. Meseguer. Vol. 4. 1996, pp. 65–89.

[Con+11] Garth Conboy et al. *EPUB 3 Overview*. Recommended Specification. International Digital Publishing Forum (IDPF), Oct. 11, 2011. URL: http://idpf.org/epub/30/spec/epub30-overview.html.

[Coq15] Coq Development Team. *The Coq Proof Assistant: Reference Manual*. Tech. rep. INRIA, 2015.

[Cra+10] Marcos Cramer et al. "The Naproche Project Controlled Natural Language Proof Checking of Mathematical Texts". In: *Controlled Natural Language, Workshop on Controlled Natural Language, CNL 2009. Revised Papers*. Ed. by Norbert E. Fuchs. LNCS 5972. Springer, 2010, pp. 170–186. ISBN: 978-3-642-14417-2. DOI: 10.1007/978-3-642-14418-9_11.

[Cre16] John Cremona. "The L-Functions and Modular Forms Database Project". In: *Foundations of Computational Mathematics* 16.6 (2016), pp. 1541–1553. ISSN: 1615-3383. DOI: 10.1007/s10208-016-9306-z.

[Dah+11] *Scalable Vector Graphics (SVG) 1.1 (Second Edition)*. W3C Recommendation. World Wide Web Consortium (W3C), Apr. 2011. URL: http://www.w3.org/TR/SVG11.

[dB08] L. de Moura and N. Bjørner. "Z3: An Efficient SMT Solver". In: *Tools and Algorithms for the Construction and Analysis of Systems*. Ed. by C. Ramakrishnan and J. Rehof. Springer, 2008, pp. 337–340.

[DB12] *Open Document Format for Office Applications (OpenDocument) v1.2*. OASIS Standard. 2012. URL: http://docs.oasis-open.org/office/v1.2/os/OpenDocument-v1.2-os.html.

[de +15] L. de Moura et al. "The Lean Theorem Prover (System Description)". In: *Automated Deduction*. Ed. by A. Felty and A. Middeldorp. Springer, 2015, pp. 378–388.

[Fau+17] Steve Faulkner et al. *HTML 5.2*. W3C Recommentation. World Wide Web Consortium (W3C), Oct. 28, 2017. URL: http://w3.org/TR/html52/.

[GSC15] Ferruccio Guidi and Claudio Sacerdoti Coen. "A Survey on Retrieval of Mathematical Knowledge". In: *Intelligent Computer Mathematics 2015*. Ed. by Manfred Kerber et al. LNCS 9150. Springer, 2015, pp. 296–315. ISBN: 978-3-319-20615-8. DOI: 10.1007/978-3-319-20615-8_20.

[Har96] J. Harrison. "HOL Light: A Tutorial Introduction". In: *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design*. Springer, 1996, pp. 265–269.

[HOL] HOL4 development team. http://hol.sourceforge.net/.

[Hor00] Helmut Horacek. "Tailoring Inference-Rich Descriptions Through Making Compromises Between Conflicting Cooperation Principles." In: *Int. J. Human-Computer Studies* 53 (2000), pp. 1117–1146.

[Hua96] Xiaorong Huang. *Human Oriented Proof Presentation: A Reconstructive Approach*. DISKI 112. Sankt Augustin, Germany: Infix, 1996.

[JFM] *Journal of Formalized Mathematics*. URL: http://www.mizar.org/JFM (visited on 09/27/2012).

[JN] *Jupyter Notebook*. URL: http://jupyter-notebook.readthedocs.org/en/latest/notebook.html#notebook-documents (visited on 08/22/2017).

[Kam+14] Fairouz Kamareddine et al. "Computerising Mathematical Text". In: *Computational Logic*. Ed. by Dov M. Gabbay, Jörg Siekmann, and John Woods. North Holland, 2014, pp. 343–396. ISBN: 9780444516244.

[KLN04] Fairouz Kamareddine, Twan Laan, and Rob Nederpelt. *A Modern Perspective on Type Theory: From its Origins until Today*. Applied Logic 29. Kluwer, 2004.

[KN04] Fairouz Kamareddine and Rob Nederpelt. "A refinement of de Bruijn's formal language of mathematics". In: *Logic, Language and Information* 13.3 (2004), pp. 287–340. URL: http://www.macs.hw.ac.uk/~fairouz/forest/papers/journals-publications/kjour.pdf.

[Knu84] Donald E. Knuth. *The TEXbook*. Addison Wesley, 1984.

[Koh06] Michael Kohlhase. *OMDoc – An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[Koh08] Michael Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. URL: https://kwarc.info/kohlhase/papers/mcs08-stex.pdf.

[Koh+11] Michael Kohlhase et al. "The Planetary System: Web 3.0 & Active Documents for STEM". In: *Procedia Computer Science* 4 (2011): *Special issue: Proceedings of the International Conference on Computational Science (ICCS)*. Ed. by Mitsuhisa Sato et al. Finalist at the Executable Paper Grand Challenge, pp. 598–607. DOI: 10.1016/j.procs.2011.04.063.

[Koh13] Michael Kohlhase. "The Flexiformalist Manifesto". In: *14th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2012)*. Ed. by Andrei Voronkov et al. Timisoara, Romania: IEEE Press, 2013, pp. 30–36. ISBN: 978-1-4673-5026-6. URL: http://kwarc.info/kohlhase/papers/synasc13.pdf.

[KU15] C. Kaliszyk and J. Urban. "HOL(y)Hammer: Online ATP Service for HOL Light". In: *Mathematics in Computer Science* 9.1 (2015), pp. 5–22.

[Lam94] Leslie Lamport. *LaTeX: A Document Preparation System, 2/e*. Addison Wesley, 1994.

[Lan30] Edmund Landau. *Grundlagen der Analysis*. Reprint of the edition, Leipzig, 1970. Darmstadt, Germany; second edition: Wissenschaftliche Buchgesellschaft, 1930.

[LM] *The L-functions and Modular Forms Database*. URL: http://www.lmfdb.org (visited on 02/01/2016).

[LTX] Bruce Miller. *LaTeXML: A LaTeX to XML Converter*. URL: http://dlmf.nist.gov/LaTeXML/ (visited on 03/12/2013).

[Mat] *MathPlayer. ¡Display MathML in your browser¿*. URL: http://www.dessci.com/en/products/mathplayer (visited on 03/19/2012).

[Mil] Dale Miller. *λProlog*. URL: http://www.lix.polytechnique.fr/Labo/Dale.Miller/lProlog/.

[Miz] *Mizar*. httpxbpap://www.mizar.org. seen 2013-02-27. URL: http://www.mizar.org.

[MML310] Ron Ausbrooks et al. *Mathematical Markup Language (MathML) Version 3.0*. Ed. by David Carlisle, Patrick Ion, and Robert Miner. 2010. URL: http://www.w3.org/TR/MathML3.

[MNB] *Wolfram Notebooks*. URL: http://www.wolfram.com/notebooks/ (visited on 03/29/2019).

[Nor05] U. Norell. *The Agda WiKi*. http://wiki.portal.chalmers.se/agda. 2005.

[OEIS] *The On-Line Encyclopedia of Integer Sequences*. URL: http://oeis.org (visited on 05/28/2017).

[OOXML06] *Standard ECMA-376 - Office Open XML File Formats*. Tech. rep. 2006. URL: http://www.ecma-international.org/publications/standards/Ecma-376.htm.

[ORS92] S. Owre, J. Rushby, and N. Shankar. "PVS: A Prototype Verification System". In: *11th International Conference on Automated Deduction (CADE)*. Ed. by D. Kapur. Springer, 1992, pp. 748–752.

[Pau94] L. Paulson. *Isabelle: A Generic Theorem Prover*. Vol. 828. Lecture Notes in Computer Science. Springer, 1994.

[Qed] *The QED Project*. http://www-unix.mcs.anl.gov/qed/. 1996. URL: http://www-unix.mcs.anl.gov/qed/.

[Ret09] Krzysztof Retel. "Gradual Computerisation and Verification of Mathematics. MathLang's Path into Mizar". PhD thesis. Edinburgh: Heriot-Watt University, Apr. 2009.

[RV02] A. Riazanov and A. Voronkov. "The design and implementation of Vampire". In: *AI Communications* 15 (2002), pp. 91–110.

[Sch01] S. Schulz. "System Abstract: E 0.61". In: *International Joint Conference on Automated Reasoning*. Ed. by R. Goré, A. Leitsch, and T. Nipkow. Springer, 2001, pp. 370–375.

[Slo03] Neil J. A. Sloane. "The On-Line Encyclopedia of Integer Sequences". In: *Notices of the AMS* 50.8 (2003), p. 912.

[sTeX] *sTeX: A semantic Extension of TeX/LaTeX*. URL: https://github.com/sLaTeX/sTeX (visited on 05/15/2015).

[W2X] *Writer2xhtml – Apache OpenOffice Extensions*. URL: https://extensions.openoffice.org/fr/project/writer2xhtml (visited on 12/01/2019).

[Wei+02] C. Weidenbach et al. "SPASS Version 2.0". In: *Conference on Automated Deduction*. Ed. by A. Voronkov. Springer, 2002, pp. 275–279.

[Wen07] Makarius Wenzel. "Isabelle/Isar — a generic framework for human-readable proof documents." In: *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*. Ed. by R. Matuszewski and A. Zalewska. Vol. 10:23. Studies in Logic, Grammar and Rhetoric. University of Białystok, 2007, pp. 277–298. URL: http://mizar.org/trybulec65/.

[Wik] Wikimedia Foundation, ed. *Wikipedia, the free encyclopedia*. URL: http://www.wikipedia.org (visited on 08/11/2010).

[WR10] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*. 2nd ed. Vol. I. Cambridge, UK: Cambridge University Press, 1910.

[Wri] Joseph Wright. ***siunitx*** – *A comprehensive (SI) units package*. URL: http://ctan.org/pkg/siunitx (visited on 12/02/2019).

[ZBF] *zbMATH – Formula Search*. URL: https://zbmath.org/formulae/ (visited on 03/28/2019).