Computerizing Mathematical Text with MathLang

Fairouz Kamareddine and J. B. Wells

School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh, Scotland
http://www.macs.hw.ac.uk/ultra/

Abstract

Mathematical texts can be computerized in many ways that capture differing amounts of the mathematical meaning. At one end, there is document imaging, which captures the arrangement of black marks on paper, while at the other end there are proof assistants (e.g., Mizar, Isabelle, Coq, etc.), which capture the full mathematical meaning and have proofs expressed in a formal foundation of mathematics. In between, there are computer typesetting systems (e.g., LaTeX and Presentation MathML) and semantically oriented systems (e.g., Content MathML, OpenMath, OMDoc, etc.).

The MathLang project was initiated in 2000 by Fairouz Kamareddine and Joe Wells with the aim of developing an approach for computerizing mathematical texts and knowledge which is flexible enough to connect the different approaches to computerization, which allows various degrees of formalization, and which is compatible with different logical frameworks (e.g., set theory, category theory, type theory, etc.) and proof systems. The approach is embodied in a computer representation, which we call MathLang, and associated software tools, which are being developed by ongoing work. Three Ph.D. students (Manuel Maarek (2002/2007), Krzysztof Retel (since 2004), and Robert Lamar (since 2006)) and over a dozen master's degree and undergraduate students have worked on MathLang. The project's progress and design choices are driven by the needs for computerizing real representative mathematical texts chosen from various branches of mathematics.

Currently, MathLang supports entry of mathematical text either in an XML format or using the TeXmacs editor. Methods are provided for adding, checking, and displaying various information *aspects*. One aspect is a kind of weak type system that assigns categories (term, statement, noun (class), adjective (class modifier), etc.) to parts of the text, deals with binding names to meanings, and checks that a kind of grammatical sense is maintained. Another aspect allows weaving together mathematical meaning and visual presentation and can associate natural language text with its mathematical meaning. Another aspect allows identifying chunks of text, marking their roles (theorem, definition, explanation, example, section, etc.), and indicating relationships between the chunks (A uses B, A contradicts B, A follows from B, etc.). Software tool support can use this aspect to check and explain the overall logical structure of a text. Further aspects are being designed to allow adding additional formality to a text such as proof structure and details of how a human-readable proof is encoded into a fully formalized version (so far this has only been done for Mizar and started for Isabelle). A number of mathematical texts have been computerized, helping with the development of these aspects, and indicating what additional work is needed for the future. This paper surveys the past and future work of the MathLang project.

Keywords: mathematical knowledge management, mathematical vernacular, mathematical typesetting, logical foundations of mathematics, proof assistants, proof checkers, theorem provers

1 Background and motivation

The MathLang project is based on considering these two questions:

(i) What is the relationship between the logical foundations of mathematical reasoning and the actual practice of mathematicians?

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

(ii) In what ways can computers support the development and communication of mathematical knowledge?

1.1 Logical Foundations

Our first question, of the relationship between the practice of mathematics and its logical foundations, has been an issue for at least two millennia. Logic was already influential in the study and development of mathematics since the time of the ancient Greeks. One of the main issues was already known by Aristotle, namely that for a logical/mathematical proposition Φ,

- given a purported proof of Φ, it is not hard to check whether the argument really proves Φ, but
- in contrast, if one is asked to find a proof of Φ, the search may take a very long time (or even go forever without success) even if Φ is true.

Aristotle used logic to reason about everything (mathematics, law, farming, medicine, etc.). A formal logical style of deductive reasoning about mathematics was introduced in Euclid's geometry [16].

The 1600s saw a increase in the importance of logic. Researchers like Leibniz wanted to use logic to address not just mathematical questions but also more esoteric questions like the existence of God. In the 1800s, the need for a more precise style in mathematics arose, because controversial results had appeared in analysis [17]. Some controversies were solved by Cauchy's precise definition of convergence in his *Cours d'Analyse* [5], others benefitted from the more exact definition of real numbers given by Dedekind [11], while at the same time Cantor was making a tremendous contribution to the formalisation of set theory and number theory [3,4] and Peano was making influential steps in formalized arithmetic [33] (albeit without an extensive treatment of logic or quantification).

In the last decades of the 1800s, the contributions of Frege made the move toward formalization much more serious. Frege found

> "*... the inadequacy of language to be an obstacle; no matter how unwieldy the expressions I was ready to accept, I was less and less able, as the relations became more and more complex, to attain precision*"

Based on this understanding of a need for greater preciseness, Frege presented *Begriffsschrift* [12], the first formalization of logic giving logical concepts via symbols rather than natural language. "Begriffsschrift" is the name both of the book and of the formal system the book presents. Frege wrote this about the Begriffsschrift:

> "*Its first purpose, therefore, is to provide us with the most reliable test of the validity of a chain of inferences and to point out every presupposition that tries to sneak in unnoticed, so that its origin can be investigated.*"

Later, Frege wrote the *Die Grundlagen der Arithmetik* and *Grundgesetze der Arithmetik* [13,14,38] where he argued that mathematics is a branch of logic and described arithmetic in the Begriffsschrift. Frege's Grundgesetze was the culmination of his work on building a formal foundation for mathematics.

One of the major issues in the logical foundations of mathematics is that the naive approach of Frege's Grundgesetze (and Cantor's earlier set theory) is inconsistent. Russell discovered a paradox in Frege's system (and also Russell's own system) that allows proving a contradiction, from which everything can be proven, including all the false statements [38,17]. The need to build logical foundations for mathematics that do not suffer from such paradoxes has led to many diverging approaches. Russell invented a form of *type theory* which he used in the famous *Principia Mathematica* [39]. Others have subsequently introduced many kinds of type theories and modern type theories are quite different from Russell's. Brouwer introduced a different direction, that of *intuitionism*. Later, ideas from intuitionism and type theory were combined, and even extended to cover the power of classical logic (which Brouwer's intuitionism rejects). Zermelo followed a different direction in introducing an axiomatization of set theory [41], later extended by Fraenkel and Skolem to form the well known Zermelo/Fraenkel (ZF) system. In yet another direction, it is possible to use category theory as a foundation. And there are other proposed foundations, too many to discuss here.

Despite the variety of possible foundations for mathematics, in practice real mathematicians do not express their work in terms of a foundation. It seems that most modern mathematicians tend to *think* in terms that are compatible with ZFC (which is ZF extended with the Axiom of Choice), but in practice they almost never write the full formal details. And it is quite rare for mathematicians to do their thinking while regarding a type theory as the foundation, even though type theories are among the most thoroughly developed logical foundations (in particular with well developed computer proof software systems). Instead, mathematicians write in a kind of *common mathematical language* (CML) (sometimes called a *mathematical vernacular*), for a number of reasons:

- Mathematicians have developed conventional ways of using nouns, adjectives, verbs, sentences, and larger chunks of text to express mathematical meaning. However, the existing logical foundations do not address the convenient use of natural language text to express mathematical meanings.
- Using a foundation requires picking one specific foundation, and any foundation commits to some number of fixed choices. Such choices include what kinds of mathematical objects to take as the primitives (e.g., sets, functions, types, categories, etc.), what kinds of logical rules to use (e.g., "natural deduction" vs. "logical deduction", whether to allow the full power of classical logic, etc.), what kinds of syntax and semantics to allow for logical propositions (first-order vs. higher-order), etc. Having made some initial choices, further choices follow, e.g., for a set theory one must then choose the axioms (Zermelo/Fraenkel, Tarski/-Grothendieck, etc.), or for a type theory the kinds of types and the typing rules (Calculus of Constructions, Martin-Löf, etc.). Fixed choices make logical foundations undesirable to use for two reasons:
  · Much of mathematics can be built on top of all of the different foundations. Hence, committing to a particular foundation would seem to unnecessarily limit the applicability of mathematical results.
  · The details of how to build some mathematical concepts can vary quite a bit from foundation to foundation. Issues that cause difficulty include how to han-

dle "partial functions", induction, reasoning modulo equations, etc. Because these issues *can* be handled in all foundations, practicing mathematicians tend to see the low-level details of these issues as inessential, irrelevant, and uninteresting, and are not willing to write the low-level details.

· Some mathematics only works for some foundations. Hence, for a mathematician to develop the specialized expertise needed to express mathematics in terms of one particular foundation would seem to unnecessarily limit the scope of mathematics the mathematician could address. An ordinary mathematician is happy to be reassured by a mathematical logician that what they are doing *can* be expressed in some foundation, but the ordinary mathematician usually does not care to work out precisely how.

Furthermore there is no universal agreement as to which is the best logical foundation.

- In practice, formalizing a mathematical text in any of the existing foundations is an extremely time-consuming, costly, and mentally painful activity. Formalization also requires special expertise in the particular foundation used that goes far beyond the ordinary expertise of even extremely good mathematicians. Furthermore, mathematical texts formalized in any of the existing foundations are generally structured in a way which is radically different from what is optimal for the human reader's understanding, and which is difficult for ordinary mathematicians to use. (Some proof software systems like Mizar (which is based on Tarski/Grothendieck set theory) attempt to reduce this problem, and partially succeed.) What is a single step in a usual human-readable mathematical text may turn into a multitude of smaller steps in a formalized version. New details completely missing from the human-readable version may need to be woven throughout the entire text. The original text may need to be reorganized and reordered so radically that it seems like it is almost turned inside out in the formal version.

So, although mathematics was a driving force for the research in logic in the 19th and 20th century, mathematics and logic have kept a distance from each other. Practicing mathematicians do not want to use formal mathematical logic and have for centuries done most mathematical work outside of the strict boundaries of formal logic.

1.2 Computerization of Mathematical Knowledge

Our second question, of how to use mechanical computers to support mathematical knowledge, is more recent but is unavoidable since automation and computation can provide tremendous services to mathematics. (There are also extensive opportunities for combining progress in logic and computerization not only in mathematics but also in other areas: program verification, bio-informatics, chemistry, music, etc.)

Mechanical computers have been used from their beginning for mathematical purposes. Starting in the 1960s, computers began to play a role in handling not just computations, but abstract mathematical knowledge. Nowadays, computers can represent mathematical knowledge in various ways:

- Pixel map images of pages of mathematical articles may be stored on the com-

puter. While useful, it is extremely difficult for computer programs to access the semantics of mathematical knowledge represented this way. Even keyword searching is difficult, because first OCR (Optical Character Recognition) must be performed and high quality OCR for mathematical material is still an area with significant research challenges rather than a proven technology (e.g., there is great difficulty with matrices [26]).

- Typesetting systems like LaTeX or TeX_{MACS} [37], can be used with mathematical texts for editing them and formatting them for viewing or printing. The document formats of these systems can also be used for storage and archiving. Such systems provide good defaults for visual appearance and allow fine control when needed. They support commonly needed document structures and allow custom structures to be created, at least to the extent of being able to produce the correct visual appearance.

  Unfortunately, unless the mathematician is amazingly disciplined, the logical structure of symbolic formulas is not directly represented. Furthermore, the logical structure of mathematics as embedded in natural language text is not represented at all. This makes it difficult for computer programs to access document semantics because fully automated discovery of the semantics of natural language text still performs too poorly to use in practical systems. Even human-assisted semi-automated semantic analysis of natural language is primitive, and we are aware of no such systems with special support for mathematical text. As a consequence, there is generally no computer support for checking the correctness of mathematics represented this way or for doing searching based on semantics (as opposed to keywords).

- Mathematical texts can be written in more semantically oriented document representations like OpenMath [1] and OMDoc [27], Content MathML [6], etc. There is generally support for converting from these representations to typesetting systems like LaTeX or Presentation MathML in order to produce readable and printable versions of the mathematical text. These systems are better than the typesetting systems at representing the knowledge in a computer-accessible way. Some aspects of the semantics of symbolic formulas can be represented in some of these systems.

  Unfortunately, in practice it is still difficult to have enough control over visual presentation with representations like OMDoc, so practicing mathematicians still prefer to use the typesetting systems.

  Systems like OMDoc share the same difficulties with accessing the logical structure of mathematics embedded in natural language text as mentioned above for typesetting systems. Although systems like OMDoc have ways to associate symbolic formulas with uninterpreted chunks of natural language text, these chunks are opaque to the computer and there is no method for checking that these associations are correct.

  A separate weakness is that although there is support for semantics, unlike proof systems (see below), OMDoc and similar systems do not have good support for expressing the semantics in terms of a logical foundation of mathematics. Also, type checking symbolic formulas (beyond mere arity checking), which is an important tool for ensuring that symbolic formulas even have meaningful seman-

tics, is not generally handled by these systems.

- There are software systems like *proof assistants* (sometimes called *proof checkers*, these include Coq, Isabelle, Mizar, Isar, etc.) and automated *theorem provers* (Boyer-Moore, Otter, etc.), which we collectively call *proof systems*. Each proof system provides a formal languages for writing mathematics based on some foundation of logic and mathematics. Work on computer support for formal foundations began in the late 1960s with work by de Bruijn on Automath (AUTOmating MATHematics) [32]. Automath supported automated checking of the full correctness of a mathematical text written in Automath's formal language. Since then, many proof systems have been built to mechanically check logic, mathematics, and computer software (e.g., Boyer-Moore, Isabelle, HOL, Coq, etc.). Generally, these systems support checking of full correctness, and it is possible in theory (although not necessarily easy) for computer programs to access and manipulate the semantics of the mathematical statements.

  Unfortunately, there are great disadvantages in using these systems. First, all of the problems mentioned for logical foundations in section 1.1 are incurred, e.g., the enormous expense of formalization. Furthermore, one must choose a specific proof system (Isabelle, Coq, Mizar, PVS, etc.) and each software system has its own advantages and pitfalls and takes quite some time to learn. In practice, some of these systems are only ever learned from a "master" in an "apprenticeship" setting. Most proof systems have no meaningful support for the mathematical use of natural language text. A notable exception is Mizar, which however requires the use of natural language in a rigid and somewhat inflexible way. Most proof systems suffer from the use of proof tactics, which make it easier to construct proofs and make proofs smaller, but obscure the reasoning for readers because the meaning of each tactic is often *ad hoc* and implementation-dependent. As a result of these and other disadvantages, ordinary mathematicians do not generally read mathematics written in the language of a proof system, and are usually not willing to spend the effort to formalize their own work in a proof system.

- Computer algebra systems (e.g., Maxima, Maple, Mathematica, etc.) are widely used software environments designed for carrying out computations, primarily symbolic but sometimes also numeric. Each CAS has a language for writing mathematical expressions and statements and for describing computations. The languages can also be used for representing mathematical knowledge. The main advantage for such a language is integration with a CAS.

  Typically, a CAS language is not tied to any specific foundation and has little or no support for guaranteeing correctness of mathematical statements. A CAS language also typically has little or no support for embedded natural language text, or for precise control over typesetting. So a CAS is often used for calculating results, but these results are usually converted into some other language or format for dissemination or verification. Nonetheless, there are useful possibilities for using a CAS for archiving and communicating mathematical knowledge.

We are gradually developing a system named MathLang which we hope will eventually be usable as a bridge between more than one of the above categories of ways of representing mathematical knowledge. We also aim for MathLang to make

easier (without requiring) the partial or full formalization of mathematical texts in some foundation.

2 An overview of MathLang

2.1 The goals of MathLang

Sections 1.1 and 1.2 described issues with the practice of mathematics: the difficulty for the normal mathematician in directly using a formal foundation, and the disadvantages of the various computer representations of mathematics. To learn how to address these issues, in 2000 we (Kamareddine and Wells) began the MathLang project to develop a new mathematical language called MathLang [1], so that texts usually written in CML (the common mathematical language, expressed either with pen and paper, or LaTeX) could be written instead in MathLang in a way that satisfies these goals:

(i) A MathLang text should support the usual features of CML: natural language text, symbolic formulas, images, document structures, control over visual presentation, etc. And the usual kind of computer support for editing such texts should be available.

(ii) It should be possible to write a MathLang text precisely in a way that is significantly less ambiguous than the corresponding CML text. A MathLang text should *somehow* support representing the text's mathematical semantics and structure. The support for semantics should cover not just individual pieces of text and symbolic formulas but also the entire document and the document's relationship to other documents (to allow building connected libraries). The degree of formality in representing the mathematical semantics should be flexible, and at least one choice of degree of formality should be both inexpensive *and* useful. There should be some automated checking of the well-formedness of the mathematical semantics.

(iii) The structure of a MathLang text should follow the structure of the corresponding CML, so that the experience of reading and writing MathLang should be close to that of reading and writing CML. This should make it easier for an author to see and have confidence that a MathLang text correctly represents their intentions. Thus, if any foundational formal systems are used in MathLang, then MathLang should somehow adapt the formal systems to the needs of the authors and readers, rather than requiring the authors and readers to adapt their thinking to fit the rigid confines of any existing foundations.

(iv) The structure of a MathLang text should make it easier to support further post-authorship computer manipulations that respect its mathematical structure and meaning. Examples include semantics-based searches, computations via computer algebra systems, extraction of proof sketches (to be completed into a full formalization in a proof system), etc.

(v) A particular important case of the previous point is that MathLang should

[1] We always named the project MathLang but initially named the proposed language NML (New Mathematical Language).

support (but not require) interfacing with proof systems so that a MathLang text can contain full formal details in some foundation and the formalization can be automatically verified.

(vi) Authoring of a MathLang text should not be significantly harder for the ordinary mathematician than authoring LaTeX. Features of MathLang that the author does not want (such as formalization in a proof system) should not require any extra effort from an author.

(vii) The design of MathLang should be compatible with (as yet undetermined) future extensions to support additional uses of mathematical knowledge. Also, the design of MathLang should make it easy to combine with existing languages (e.g., OMDoc, TEX_{MACS}). In this sense, MathLang might end up being a method for extending an existing language in addition to (or possibly instead of) a language on its own.

None of the previously existing representations for mathematical texts satisfies our goals, so we have been developing new techniques.

MathLang is intended to support different degrees of formalization. Furthermore, for those documents where full formalization is a goal, MathLang is intended to allow this to be accomplished in gradual steps. Some of the motivations for varying degrees of formalization have already been discussed in sections 1.1 and 1.2. Full formalization is sometimes desirable, but also is often undesirable due to its expense and the requirement to commit to many inessential foundational details. Partial but not full formalization can sometimes be desirable for various reasons; as examples, it has the potential to be helpful with automated checking, semantics-based searching and querying, and interfacing with computer algebra systems (and other mathematical computation environments). Partial formalization can be carried out to different degrees:

- The abstract syntax trees of symbolic formulas can be represented accurately. This is usually missing when using systems like LaTeX or Presentation MathML, while more semantically oriented systems usually provide this to some degree. This can be used to provide editing support for algebraic rearrangements and simplifications, and can help with interfacing with computer algebra systems.
- The mathematical structure of natural language text can be represented in a way similar to how symbolic formulas are handled. Furthermore, mixed text and symbols can be handled. This can help in the same way as capturing the structure of symbolic formulas can help. Nearly all previous systems do not support handling natural language text in this way.
- A weak type system can be used to check simple grammatical conditions without checking full semantic sensibility.
- Justifications (inside proofs and between formal statements) can be linked (without necessarily always indicating precisely how they are used). Some examples of potential uses of this feature include the following:
  · Extracting only those parts of a document that are relevant to specific results. (This could be useful in educational systems.)
  · Checking that each instance of apparently circular reasoning is actually handled

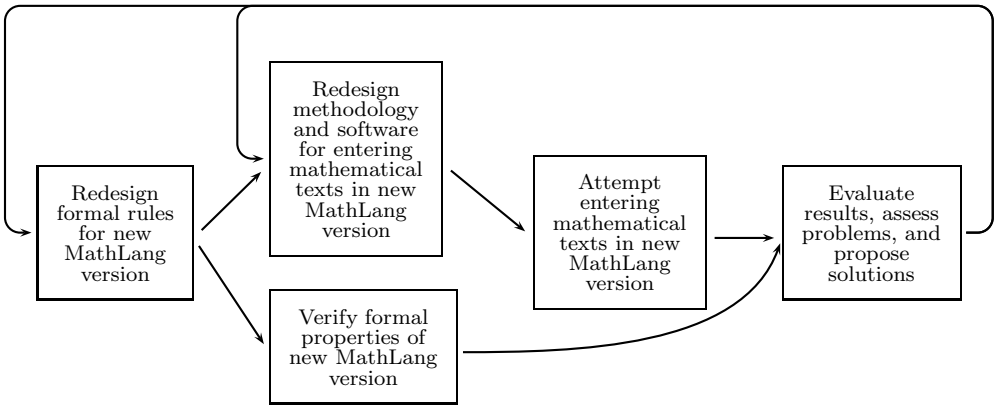

Fig. 1. Iterative MathLang development process

via induction.

· Calculating proof gaps as a first step toward fuller formalization.

- If one commits to a foundation (or in some cases, to a family of foundations), one can start to use more sophisticated type systems in formulas and statements for checking more aspects of well-formedness.
- And there are further possibilities.

2.2 The process of designing MathLang

We are gradually refining the design of MathLang based on experience testing the use of MathLang for representative mathematical texts. This iterative process is depicted in figure 1. We started from an initial design based on ideas from WTT (see below). Each iteration of the overall procedure is to test the design by evaluating encodings of real mathematical texts, during which issues and difficulties are encountered, which lead to new needs being discovered and corresponding design adjustments. The design includes not just formal rules for the representation of mathematical texts, but also patterns and methodology for entering texts in this representation, and supporting software.

Our choice of mathematical texts for testing is primarily oriented toward texts that represent the variety of mathematical writing by ordinary mathematicians rather than texts that represent the interests of formalists and mathematical logicians. Much of our testing has been with pre-existing texts. In some cases, we have chosen texts that have previously been formalized by others so that we can compare the representations, e.g., *A Compendium of Continuous Lattices* [15] of which at least 60% has been formalized in Mizar [34], and Landau's *Foundations of Analysis* [29] which was fully formalized in Automath [36]. In some other cases, we have chosen texts of historical importance which are known to have errors to ensure that MathLang's design will not exclude them, e.g., Euclid's *Elements* [16]. And we have chosen other texts to exercise other aspects of MathLang. In addition, we have been testing the authoring of new texts.

2.3 The original starting point: WTT

For purely historical reasons it is interesting to point out that the initial design of MathLang was based heavily on ideas from the Weak Type Theory (WTT) of Nederpelt and Kamareddine [25], which in turn was heavily inspired by the Mathematical Vernacular (MV) of de Bruijn [10].

In the terminology of WTT, a document is a *book* which is a sequence of *lines*, each of which is a pair of a *sentence* (a *statement* or a *definition*) and a *context* of facts (*declarations* or *statements*) assumed in the sentence. WTT has four ways of introducing names. A *definition* introduces a name whose scope is the rest of the book and associates the name with its meaning. A name introduced by a definition can have *parameters* whose scope is the body of the definition. A *declaration* in a context introduces a name (with no parameters) whose scope is only the current line. Finally, a *preface* gives names whose scope is the document; names introduced by prefaces have parameters but unlike definitions their meanings are not provided (and thus presumed to be given externally to the document). Declarations, definitions, and statements can contain *phrases* which are built from *terms*, *sets*, *nouns*, and *adjectives*. Using the terminology of object-oriented programming languages, nouns act like *classes* and adjectives act like *mixins* (a special kind of function from classes to classes). WTT uses a weak type system with types like `noun`, `set`, `term`, `adjective`, `statement definition`, `context`, and `book` to check some basic well-formedness conditions. Sets are used when something is definitely known to be a set and the richer structure of a noun is not needed, and terms are used for things that are not sets (and sometimes for sets in cases where the type system is too weak).

Although WTT provides many useful ideas, the definition of WTT has many limitations. The many different ways of introducing names are too complicated and awkward. WTT provides no way to indicate which statements are used to justify other statements and in general does not deal with *proofs* and logical correctness. WTT provides no ways to present the structure of a text to human readers; there is no way of grouping statements and identifying their mathematical/discourse roles such as *theorem*, *lemma*, *conjecture*, *proof*, *section*, *chapter*. WTT provides no way to give human names to statements (e.g., "Newman's Lemma"). WTT provides no way to use in one document concepts defined in another document.

2.4 The current MathLang design

The current MathLang design has developed through the experience of a large number of students, including both shorter projects (over a dozen projects by either 4th year undergraduate students or M.Sc. students) and Ph.D. studies (by 3 students: Maarek, Retel, and Lamar). Every student has done work to write in MathLang some piece of mathematical text. The experience gained from this has led to the current design of MathLang which is (currently) divided into three *aspects*:

- The Core Grammatical aspect (CGa) [23,24,30,19] takes the best features of WTT [25] and MV [10], simplifies difficult aspects of WTT, and enhances the nouns and adjectives of WTT with ideas from object-oriented programming so that nouns are more like classes and adjectives are more like mixins. In CGa,

the different kinds of name-introducing forms of WTT are unified; all definitions by default have indefinite forward scope and a local scope operator is used to allow local definitions. The basic unit becomes the *step*, which can be either a definition, a statement (a phrase that asserts something), or a *block* which is merely a grouping of steps. CGa keeps WTT's notions of nouns, adjectives, terms, sets, definitions, and statements. CGa provides a kind of *grammar* for well-formed mathematics with grammatical categories and allows checking for some basic well-formedness conditions (e.g., the origin of all names and symbols can be tracked).

- The Text and Symbol aspect (TSa) [22,18,30,19] allows integrating normal typesetting and authoring software with the mathematical structure represented with CGa. TSa allows weaving together usual mathematical authoring representations such as LaTeX, XML, or TeX$_{\text{MACS}}$ with CGa data. Thanks to a notion of *souring* rules (called "souring" because it does the opposite of what is usually called *syntactic sugar*), TSa allows the structure of the mathematical text to follow the structure of the CML as conceived by the mathematician.
- The Document Rhetorical aspect (DRa) [21] supports identifying portions of a text and expressing the relationships between them. Any portion of text (e.g., a phrase, a step, a block, etc.) can be given an identity. Many kinds of relationships can be expressed between identified pieces of text. For example, a chunk of text can be identified as a "theorem", and another can be identified as the "proof" of that theorem. Similarly, one chunk of text can be a "subsection" or "chapter" of another. Given these identified relationships, it becomes possible to do computations to check whether all dependencies are identified, to check whether the relationships are sensible or possibly problematic (and whether therefore the author should be warned), and to extract and explain the logical structure of a text. Dependencies identified this way have been used in generating formal proof sketches and identifying the proof holes that remain to be filled.

In addition to the design of MathLang itself, there has been work on relating a MathLang text to a fully formalized version of the text. Using the information in the CGa and DRa aspects of a MathLang text, we have developed a procedure for producing a corresponding Mizar document, first as a proof sketch with holes and then as a fully completed proof [20]. We have recently begun to work also on doing this with Isabelle in addition to Mizar.

Figure 2 (taken from [20]) diagrams the overall current situation of work on MathLang. This figure refers to Mizar because this is the only proof system we have completed documents with. In the rest of this paper, we discuss the aspects CGa, TSa, and DRa in more detail, and also discuss the work on interfacing MathLang with proof systems such as Mizar.

3 The aspects of MathLang

3.1 The Core Grammatical aspect (CGa)

CGa [23,24,30,19] is a formal language inspired initially by WTT [25] and MV [10], and then later shaped by repeated experiences of entering mathematical texts in

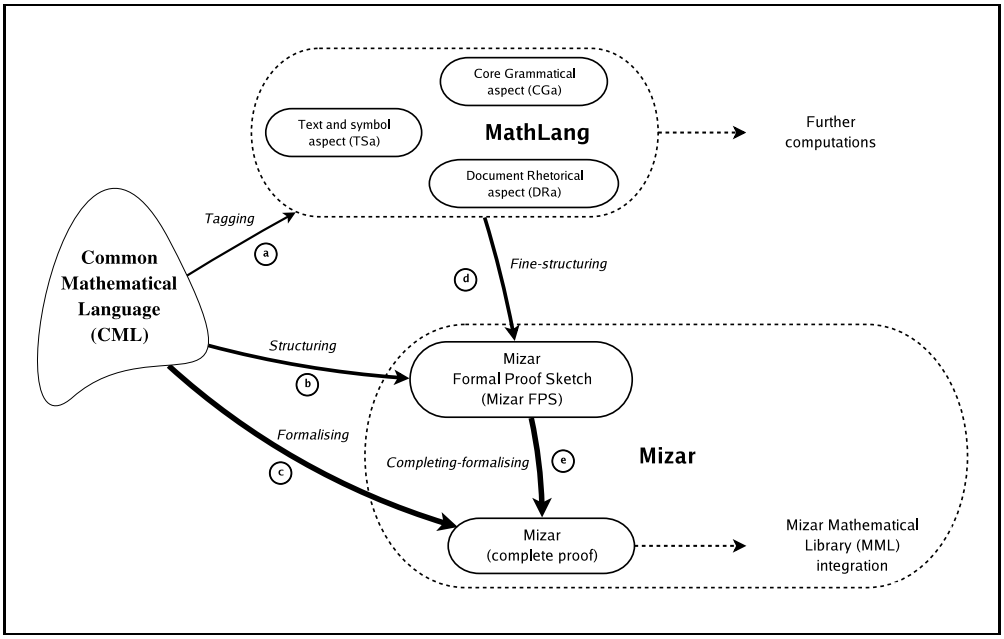

Fig. 2. Overall situation of work in MathLang

early versions of CGa.

The basic constructs of CGa are the *step* and the *expression*. The tasks handled in WTT by books, prefaces, lines, declarations, definitions, and statements are all represented as steps in CGa. A step can be a *block* $\{s_1, \ldots, s_n\}$, which is merely a sequence of steps. A step can be a *local scoping* $s_1 \triangleright s_2$, which is a pair of steps s_1 and s_2 where the definitions and declarations of s_1 are restricted in scope to s_2 and the assertions of s_1 are assumptions of s_2. A step can also be a *definition*, a *declaration*, or an expression (which asserts a truth). Expressions are also used for the bodies of definitions and inside the types in declarations. The possibilities for expressions include uses of defined identifiers, identifier declarations, and *noun descriptions*. A noun description allows specifying *characteristics* of a class of entities.

Here is an example. Consider this (silly) CML text:

> "Given that $\mathfrak{M}$ is a set, y and x are natural numbers, and x belongs to $\mathfrak{M}$, it holds that $x + y = y + x$."

A straightforward encoding of the above text in CGa would be the following:

$$\{\mathtt{M} : \mathtt{set};\ \mathtt{y} : \mathtt{natural\_number};\ \mathtt{x} : \mathtt{natural\_number};\ \in(\mathtt{x}, \mathtt{M})\}$$
$$\triangleright\ =(+(\mathtt{x}, \mathtt{y}), +(\mathtt{y}, \mathtt{x}))$$

This example assumes that somewhere earlier in the document there will be declarations like these:

$$\ldots;\ \in(\mathtt{term}, \mathtt{set}) : \mathtt{stat};\ =(\mathtt{term}, \mathtt{term}) : \mathtt{stat};\ \mathtt{natural\_number} : \mathtt{noun};$$
$$+\,(\mathtt{natural\_number}, \mathtt{natural\_number}) : \mathtt{natural\_number};\ \ldots$$

Here, `M`, `y`, `x`, $\in$, $=$, and $+$ are *identifiers*[2] while `term`, `set`, `stat`, and `noun` are *keywords* of CGa. The semicolon, colon, comma, parentheses, braces, and right triangle ($\triangleright$) symbols are part of the syntax of CGa. The statements like $\in$(`term`, `set`) : `stat` are declarations; this example declares $\in$ to be an operator that takes two arguments, one of type `term` and one of type `set`, and yields a result of type `stat` (statement). The statement `M` : `set` is an abbreviation for `M`() : `set` which declares the identifier `M` to have zero parameters.

CGa uses grammatical/linguistic/syntactic *categories* (also called *types*) to make explicit the grammatical role played by the elements of a mathematical text. In the above example, we see the category expressions `term`, `set`, `stat`, `noun`, and `natural_number`. In fact, the category expression `natural_number` acts as an abbreviation for `term`(`natural_number`), and `term`, `set`, and `noun` are abbreviations for `term`(`Noun` {}), `set`(`Noun` {}), and `noun`(`Noun` {}), which all use the *uncharacterized* noun description `Noun` {}. A noun description is of the form `Noun` s and describes a class of entities with *characteristics* (declared operations and true facts) defined by the step s. The arguments of the category constructors `term`, `set`, and `noun` are expressions which evaluate to noun descriptions. The category `term`(e) describes individual entities belonging to the class described by the noun expression e, and the category `set`(e) describes any set of such entities. The category `noun`(e) describes any noun which defines all the operations described by e with the same types. So in the above example, the abbreviation `term` is the type of all mathematical entities, the abbreviation `set` is the type of any set, `noun` is the type of any noun (and specifies no characteristics for it), and `natural_number` is the type of any mathematical entity having the characteristics described by the noun `natural_number`.[3] The behavior of nouns in CGa is similar to that of *classes* in object-oriented programming languages. CGa also has *adjectives* which are like object-oriented *mixins* and act as functions from nouns to nouns.

Here are some further examples of categories (see also [25]), where we put parts that do not determine the indicated category inside boxes:

Terms: the triangle ABC; the center of $\boxed{\text{ABC}}$; $d(\boxed{\text{x}}, \boxed{\text{y}})$.

Nouns: a triangle; an edge of $\boxed{\text{ABC}}$; a group.

Adjectives: equilateral $\boxed{\text{triangle}}$; prime $\boxed{\text{number}}$; Abelian $\boxed{\text{group}}$.

Statements: $\boxed{\text{P}}$ lies between $\boxed{\text{Q}}$ and $\boxed{\text{R}}$; $\boxed{5} \geq \boxed{3}$; $\boxed{\text{AB}}$ is $\boxed{\text{an edge of ABC}}$.

Definition: a number p is prime whenever $\boxed{\cdots}$.

These categories are all inspired from WTT. Full details of the rules of CGa are in other papers [24,30].

The types of CGa are more sophisticated than the *weak types* of WTT and allow tracking which operations are meaningful in some additional cases. Although CGa's types are more powerful than WTT's, there are still significant limitations. One limitation is that higher-order types are not allowed. For example, although CGa allows the type (`term`, `term`) $\rightarrow$ `term`, which is the type of an operator that takes two arguments of type `term` and returns a result of type `term`, CGa does not

[2] Our current implementation only allows ASCII characters in identifiers, but we plan to support any graphic Unicode characters.

[3] CGa has other mechanisms that allow specifying additional characteristics of the noun `natural_number` separate from its declaration, and we assume in this example that this is done.

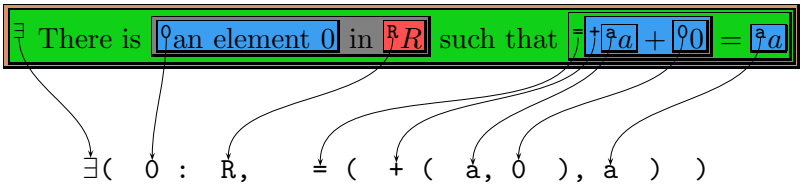

Fig. 3. Example of CGa encoding of CML text

allow using the type $((\texttt{term}) \to \texttt{term}, \texttt{term}) \to \texttt{term}$, which would be the type of an operator that takes another operator as its first argument. Higher-order types can be awkwardly and crudely emulated in CGa by encapsulation with noun types, but this emulation does not work well due to the fact that CGa's type polymorphism is shallow, which is another significant limitation. To work around the weakness of CGa's type polymorphism, in practice we find ourselves often giving entities the type `term` instead of a more precise type. We continue to work on making the type system more flexible without making it too complex. It is important to understand that the goal of CGa's type system is *not* to ensure full correctness, but merely to check whether the reasoning parts of a document are coherently built in a sensible way.

The design of CGa is due to Kamareddine, Maarek and Wells [24]. The implementation of CGa is due to Maarek [30].

3.2 The Text and Symbol aspect (TSa)

TSa [22,18,30,19] is a representation that allows interleaving pieces of CGa with pieces of CML in the form of mixtures of natural language, symbolic formulas, and formatting instructions for visual presentation. The interleaving can be at any level of granularity: meanings can be associated at a coarse grain with entire paragraphs or sections, or at a fine grain with individual words, phrases, and symbols. Arbitrary amounts of mathematically uninterpreted text can be included. The TSa representation is inspired by the XQuery/XPath Data Model (XDM) [8] used for representing the information content of XML documents. In TSa, a document d is built from the empty document ($[\,]$) by sequencing (d_1, d_2) and labeling ($\ell\langle d\rangle$).

As an example of TSa, consider the piece of CML text and its CGa representation given in figure 3.[4] The example could be represented in TSa by the following fine-grained interleaving of CGa[5] and LaTeX:

```
“There is #1 such that #2.”
  ⟨∃⟨“#1 in #2”⟨:⟨“an element $0$”⟨0⟩, “$R$”⟨R⟩⟩⟩,
     “$#1 = #2$”⟨=⟨“#1 + #2”⟨+⟨“a”⟨a⟩, “0”⟨0⟩⟩⟩, “a”⟨a⟩⟩⟩⟩
```

This example uses the abbreviation that ℓ stands for $\ell\langle[\,]\rangle$. For example, “a”⟨a⟩ actually stands for “a”⟨a⟨$[\,]$⟩⟩.

Associated with TSa are methods for extracting separately the CGa and the typesetting instructions or other visual representation. For the example, from the

[4] This example comes from [18].

[5] The representation shown here omits type/category annotations that we usually include with the CGa identifiers used in the TSa representation.

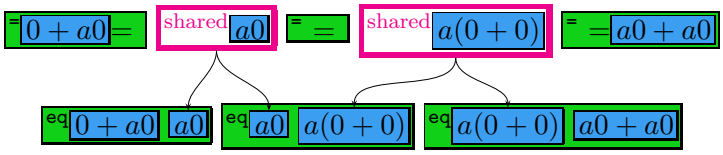

Fig. 4. Example of using souring in TSa to support sharing

TSa above can be extracted the following TSa representation of just the CGa portion:

$$\exists\langle{:}\langle 0, \mathtt{R}\rangle, {=}\langle{+}\langle \mathtt{a}, 0\rangle, \mathtt{a}\rangle\rangle$$

The CGa portion of this text can be type checked and used for processing that needs to know the mathematical meaning of the text. Similarly, the following pieces of LaTeX can also be extracted:

```
"There is #1 such that #2."
  ⟨"#1 in #2"⟨"an element $0$", "$R$"⟩,
   "$#1 = #2$"⟨"#1 + #2"⟨"a", "0"⟩, "a"⟩⟩
```

This tree of LaTeX typesetting instructions can be further flattened for actual processing by LaTeX into a string such as:

```
"There is an element $0$ in $R$ such that $a + 0 = a$."
```

The idea of the TSa representation is independent of the visual formatting language used. Although we use LaTeX in our example here, in our implementations so far we have used the TeX_{MACS} internal representation and also XML.

As part of the task of using TSa to interleave CGa and more traditional natural language and typesetting information, we have needed to develop techniques for handling certain challenging CML formations where the mathematical structure and the CML representation do not nicely match. For example, in the text $0+a0 = a0 = a(0+0) = a0+a0$, the terms $a0$ and $a(0+0)$ are each shared between two equations. Most formal representations would require either duplicating these shared terms, like for example $0 + a0 = a0 \wedge a0 = a(0 + 0) \wedge a(0 + 0) = a0 + a0$, or explicitly abstracting the shared terms. To allow the TSa representation to be as close to CML as possible, we instead solve this issue by using "*souring*" annotations in the TSa representation [18]. These annotations are a third kind of node label used in TSa, in addition to the CGa and formatting labels. We have developed methods using souring annotations for extracting both the correct mathematical meaning and the nice visual presentation in the CML style. For the above example this is depicted in figure 4.

We have developed more sophisticated annotations that can handle more complicated cases of sharing of terms between equations. Souring annotations have also been developed to support several other common CML formulations. Support for folding and mapping over lists allows using forms like $\forall a, b, c \in S.P$ as shorthand for $\forall a \in S.\forall b \in S.\forall c \in S.P$ and $\{a, b, c\}$ as shorthand for $\{a\} \cup (\{b\} \cup (\{c\} \cup \emptyset))$. We have not yet developed folding that is sophisticated enough to handle ellipsis (...)

as in CML formulations like the following example (from [35]):

$$f[\underbrace{x,\ldots,x}_{n+1 \text{ arguments}}] = \frac{f^{(n)}(x)}{n!}$$

We have implemented a user interface as an extension of the TEX$_{\text{MACS}}$ editor for entering the TSa MathLang representation. The author can use mouse and keyboard commands to annotate CML text entered in TEX$_{\text{MACS}}$ with boxes representing the CGa grammatical categories in order to assign CGa identifiers and thereby explicitly indicate mathematical meanings. The user interface allows displaying either a pure CML view which hides the TSa and CGa information, a pure CGa view, or various combined views including a view like the one depicted in figure 3. The same interface allows adding souring annotations like those depicted in figure 4.

In future work, we would like to develop techniques for not just pairing a single CML presentation with its CGa meaning, but also allowing multiple parallel visual presentations such as multiple natural languages (not just English), both natural language and symbolic formula presentations, and presentations in different symbolic notations. We would like to develop better software support to aid in semi-automatically converting existing CML texts into MathLang via TSa and CGa.

The design of TSa is due to Kamareddine, Maarek, and Wells with contributions by Lamar to the souring rules [18,30]. The implementation is primarily by Maarek [30].

3.3 The Document Rhetorical aspect (DRa)

DRa [21,20] is a system for attaching annotations to mathematical documents that indicate the roles played by different parts of a document. DRa assumes the underlying mathematical representation (which can be the MathLang aspects CGa or TSa) has some mechanism for identifying document parts.

Some DRa annotations can be unary predicates on parts; these include annotations indicating ordinary document sectioning roles such as *part*, *chapter*, *section*, etc. (like the sectioning supported by LaTeX, OMDoc, DocBook, etc.) and others indicating special mathematical roles such as *theorem*, *lemma*, *proof*, etc. Document parts can have multiple annotations if appropriate.

Other DRa annotations can be binary predicates on parts; these include such relationships between parts as "*justified by*", "*uses*", "*part of*", and "*example of*". Regarding the annotation of justifications, remember that a CML text is usually incomplete: a mathematical thought process makes jumps from one interesting point to the next, skipping over details. This does not mean that many mistakes can occur; these details are usually so obvious for the mathematician that a couple of words are enough (e.g., "*apply theorem 35*"). The mathematician knows that too many details hinder concentration. To allow MathLang text to be close to CML text, DRa allows informal justifications, which can be seen as *hints* about which statements would be used in the proof of another statement.

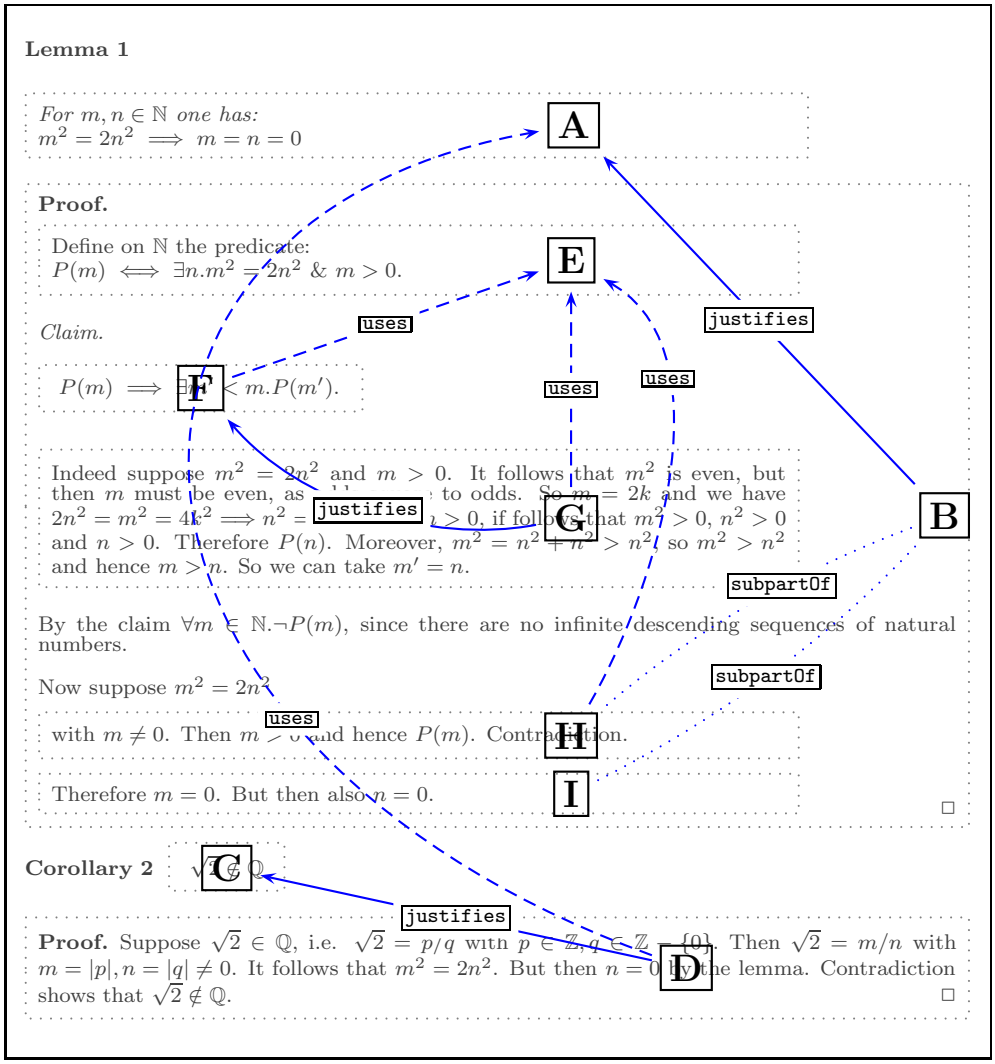

Fig. 5. Example of wrapping/naming chunks of text and marking relationships in DRa

$(A$, hasMathematicalRhetoricalRole, lemma$)$	$(B$, justifies, $A)$
$(E$, hasMathematicalRhetoricalRole, definition$)$	$(D$, justifies, $C)$
$(F$, hasMathematicalRhetoricalRole, claim$)$	$(D$, uses, $A)$
$(G$, hasMathematicalRhetoricalRole, proof$)$	$(G$, uses, $E)$
$(B$, hasMathematicalRhetoricalRole, proof$)$	$(F$, uses, $E)$
$(H$, hasOtherMathematicalRhetoricalRole, case$)$	$(H$, uses, $E)$
$(I$, hasOtherMathematicalRhetoricalRole, case$)$	$(H$, subpartOf, $B)$
$(C$, hasMathematicalRhetoricalRole, corollary$)$	$(H$, subpartOf, $I)$
$(D$, hasMathematicalRhetoricalRole, proof$)$	

Fig. 6. Example of DRa relationships between chunks of text in figure 5

Figure 5 gives an example (taken from [20] and implemented by Retel) where the mathematician has identified parts of the text (indicated by letters A through I in the figure). Figure 6, shows the underlying mathematical representation of some example DRa annotations for the example in figure 5. Here, the mathematician has given each identified part a structural (e.g., chapter, section, etc.) and/or mathe-

matical (e.g., lemma, corollary, proof, etc.) rhetorical role, and has indicated the relation between wrapped chunks of texts (e.g., justifies, uses, etc.). Note that all the DRa annotations are represented as triples; this allows using the machinery of RDF [7] (a W3C standard that is aimed at the "semantic web") to represent and manipulate them.

The DRa implementation can automatically extract a dependency graph (as seen in figure 5) that represents knowledge about how the parts of a document are related. The dependency graph can be used to check whether the logical reasoning of the text is coherent and consistent (e.g., no loops in the reasoning, except when supported by induction).

Future work with DRa will include more experience-driven tests on real CML texts, improvement of features for using the DRa structure of a text to checking it is sensibly constructed, better integration with TSa, and better support for recording different kinds of informal justifications.

The design of DRa is due to Kamareddine, Retel, and Wells with contributions by Maarek [21]. The implementation is primarily due to Retel.

4 Connecting MathLang to formal foundations

4.1 Goals for formalization

Current approaches to formalizing CML texts generally involve rewriting the text from scratch; there is no clear methodology in which the text can gradually change in small steps into its formal version. One of MathLang's goals is to support formalizing a text in small steps that do not require radically reorganizing the text. Also, a text with fully formal content should continue to be able to be presented in the same way as a less formal version originally developed by a mathematician. We envision formalization as working by adding additional layers of information to a MathLang document to support embedding formal proofs. Ideally, there should be flexible control over how much of the additional information is presented to the reader; the additional information could form part of the visual presentation, or could exist "behind the scenes" to provide assurance of correctness.

As part of the goal of supporting formalization in MathLang, we desire to keep MathLang independent of any particular formal foundation. However, as proofs embedded in a MathLang document become more formal, it will be necessary to tie them more closely to a particular proof system. It might be possible that fully formal documents could be kept independent of any particular foundation by allowing the most formal parts of a document to be expressed redundantly in multiple proof systems. (This is similar in spirit to the way the natural language portion of a document might be expressed simultaneously in multiple natural languages.)

Following the general MathLang development/design strategy in figure 1, we have been developing methodology and software for connecting a MathLang document with formal versions of its content.

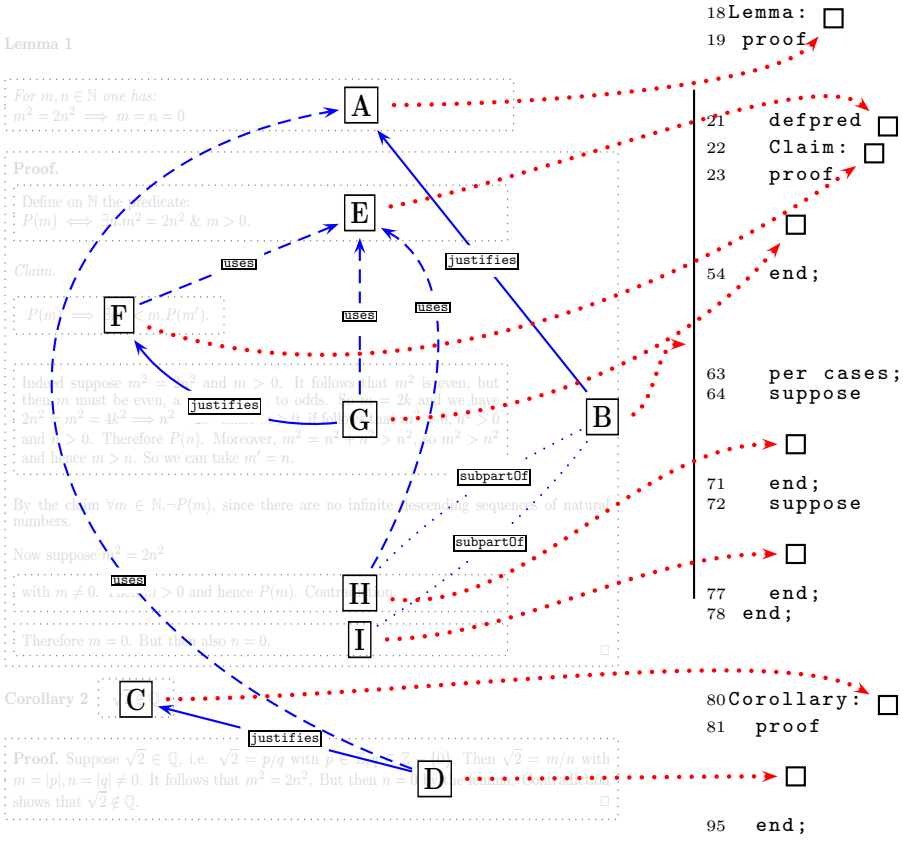

Fig. 7. Example of generating a Mizar Text-Proper skeleton from MathLang DRa and CGa

4.2 Initial exploratory work using Mizar

When the MathLang project started, we planned to start with Coq for our initial development of support for formalization, but because Krzysztof Retel joined us with his previous Mizar experience, we started with Mizar instead. Our work so far with Retel on extending MathLang documents into Mizar formalizations [20] involves constructing a skeleton of a Mizar document from a MathLang document, and then completing the Mizar skeleton separately. A Mizar document consists of an Environment-Declaration and a Text-Proper. In Mizar, the Environment-Declaration is used to generate the Environment which has the needed knowledge from MML (Mizar's Mathematical Library). The Text-Proper is checked for correctness using the knowledge in the Environment.

Our work on creating a Mizar document skeleton from a MathLang document uses the information in the CGa and DRa MathLang aspects. As an example, a DRa relationship $(B, \mathsf{justifies}, A)$ requires that some portion of the Text-Proper portion of the Mizar skeleton must look like this, where E and D must be filled in appropriately (and the word "theorem" may need to be "lemma" or similar instead,

as determined by the role of A):

```
Theorem:
E
proof
D
end;
```

Each DRa relationship and each portion of the CGa information contributes one or more constraints toward the formation of the Mizar skeleton. Assuming these constraints are solvable (which will usually be the case unless the mathematics is horribly wrong), a Mizar skeleton is produced. If enough information is present, the Mizar skeleton qualifies as a Mizar Formal Proof Sketch (FPS) [40]. Figure 7 (taken from [20]) illustrates this process for our example document given in figures 5 and 6. Given a Mizar FPS, a Mizar expert can complete the FPS by filling in the remaining gaps in the reasoning.

4.3 Toward practicality and multiple proof systems

There is much work still left to be done so that connecting MathLang to a foundational proof system can be practical. For the work we have done so far with Mizar, once the Mizar skeleton has been generated, completing the formalization is done in Mizar. It would fit the goals of MathLang better if the information needed to complete a formal proof in some proof system were instead stored in the MathLang document and manipulated using the same software tools that are used for editing MathLang documents. It is desired that MathLang texts can contain enough information to generate fully formal proofs (or equivalent proof scripts) that can be verified without further human interaction by a proof system.

The precise details of how to include this information in a MathLang text are still being designed. The current design of CGa and DRa is deliberately far too weak to fully formalize a text in any mathematical foundation, and TSa is solely concerned with connecting mathematical content to its visual appearance. Because we want to keep the core mathematical aspects of MathLang (currently these are CGa and DRa) independent of particular foundations, in these aspects some mathematical tools (e.g., induction, partial functions, etc.) are best treated as "black boxes" because they are formalized differently in different foundations. Also, it is intended that many aspects of a MathLang text will be largely independent of the choice of proof system (like Mizar) or mathematical foundation (like the Tarski/Grothendieck set theory used by Mizar), so that MathLang texts will be reusable in more situations. We expect that embedding formal proof content will require some combination of strengthening CGa and DRa and adding one or more additional MathLang aspects.

We do not yet know the full requirements of what extensions will be needed. The ideas in Barendregt's MPL (Mathematical Proof Language) [2] may be useful. We expect to investigate whether a system like Automath can be used conveniently as a meta-system for encoding other proof systems. Notions like the Mizar concepts of "Environment" and "Text-Proper" are needed, which can then be easily transformed into the format of a real proof system like Mizar FPS, while at the same time remaining as independent of any proof system as possible. Special support may be

needed for *holes* as part of incremental proof completion, because the meta-theory of holes for some proof systems remains problematic. A question that we will be continually asking is where commitment to a particular foundation or proof system is needed and where such commitment can be avoided. An overriding concern is that the structure of a MathLang text should correspond to the *human* conception, rather than the contortions usually required by a foundation.

Our self evaluation will continue to use the test documents we have already identified and may in the future involve choosing additional CML documents. When complete, we will compare the resulting formalizations in MathLang for *A Compendium of Continuous Lattices* [15] and Landau's *Foundations of Analysis* [29] with the already-existing fully proof-checked formalizations in Mizar and Automath. We also plan to evaluate MathLang using additional foundations. We have started using Isabelle (due to Robert Lamar's previous Isabelle experience) in work that is similar to the work done with Mizar. Another foundation we also plan to target is the Calculus of Constructions (CoC) [9], which is the core of what is implemented by the Coq proof system.

5 Conclusion

5.1 Work and accomplishments so far

The work on MathLang has been led by Kamareddine and Wells, has involved the hard work of 3 Ph.D. students (Maarek, Retel, and Lamar), and has benefited from implementation and evaluation work by numerous undergraduate and master's degree students. This work has led to a number of publications [23,22,24,18,19,20,21] and Maarek's Ph.D. thesis [30]. Retel's Ph.D. thesis is expected imminently, while Lamar's Ph.D. studies are nearing the halfway point. A new Ph.D. student Christoph Zengler is starting soon.

To compare our initial plans and our actual achievements, figure 8 contains a project planning dependency diagram from an early MathLang project plan that shows the tasks and the dependencies between them as we imagined them in 2001.[6] Six years later, we can see that the development of CGa [23,24,30,19], TSa [22,18,30,19], and DRa [21,20] represent substantial progress on tasks `1`, `2`, `3`, and `4`, although more remains to be done on these tasks. The work on learning how to use CGa and DRa to generate Mizar proofs represents partial progress on task `7`. All of our work so far has included work on parts of task `9`.

We have not yet worked on task `5`, which envisioned altering the structure of a specific mathematical foundation to make it more suitable for embedding in ordinary mathematical documents. We might still do this, but we are now more likely to work on techniques that can be used with more than one foundation and that can avoid altering the foundation. Also, we have not used for MathLang a proof system based on CoC. Although we are currently working on connecting MathLang documents with formalizations in Mizar and Isabelle, we plan to connect with at least one more proof system, most likely Coq (which is based on CoC).

[6] The only change from our original diagram is that in addition to calling the project "MathLang" (as we always have), we now also call the language "MathLang" instead of the older name of "NML" (New Mathematical Language).

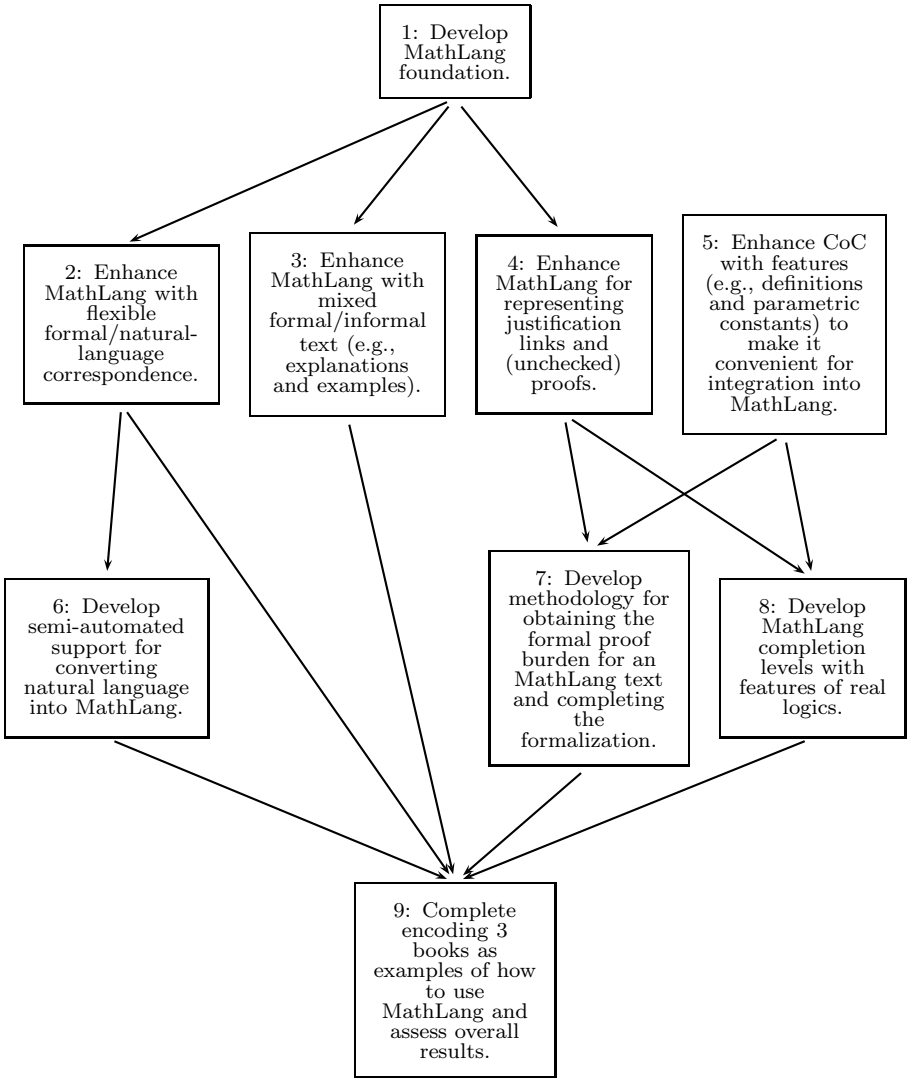

Fig. 8. MathLang project planning diagram from the year 2001

Our initial plans for task 8 have replaced a concept we called "completion levels" by our notion of "aspects". The semi-automatic extraction of mathematical structure from CML proposed in task 6 remains for future work.

5.2 Future work and planned results

MathLang is a long-term project and we expect there will be years of design, implementation, and evaluation, followed by repeated redesign, reimplementation, and re-evaluation. There are many areas which we have identified as needing more work and investigation. One area is improvements to the MathLang software (currently based on the TEX_{MACS} editor) to make it easier to enter information for the core MathLang aspects (currently CGa and DRa). This is likely to include work on semi-automatically recognizing the mathematical meaning of natural language text. A second area is further designing and developing the portions of MathLang needed for better support of formalization. An issue here is how much expertise in any

particular target proof system will be needed for authoring. It may be possible to arrange things in MathLang to make it easy for an expert in a proof system to collaborate with an ordinary mathematician in completing a formalization. A third area where work is needed is in the overall evaluation process needed to ensure MathLang meets actual needs. This will require testing MathLang with ordinary mathematicians, mathematics students, and other users. And there are additional areas where work will be needed, including areas we have not yet anticipated.

The MathLang project aims for a number of outcomes. MathLang aims to support mathematics as practiced by the ordinary mathematician, which is generally not formalized, as well as work toward full formalization. We expect that after further improvements on the MathLang design and software, writing MathLang documents (without formalizing them) will be easy for ordinary mathematicians. MathLang will support various kinds of consistency checking even for non-formalized mathematics. MathLang will be independent of any particular logical foundation of mathematics; individual documents will be able to be formal in one or more particular foundations, or not formalized.

MathLang hopes to open a new useful era of collaboration between ordinary mathematicians, logicians (who ordinarily stay apart from other mathematicians), and computer science researchers working in such areas as theorem proving and mathematical knowledge management who can develop tools to link them together. The MathLang project's outputs will include a document representation, software suitable for manipulating this representation, and documentation and guidance for how to use the representation and the software. MathLang's document representation is intended to help with various kinds of automated computerized processing of mathematical knowledge. It should be possible to link MathLang documents together to form a public library of reusable mathematics. MathLang aims to better support translation between natural languages of mathematical texts and multilingual texts. MathLang aims to better support the differing uses of mathematical knowledge by different kinds of people, including ordinary practicing mathematicians, students, computer scientists, logicians, linguists, etc.

References

[1] J. Abbott, A. van Leeuwen, and A. Strotmann. Objectives of openmath. Technical Report 12, RIACA (Research Institute for Applications of Computer Algebra), 1996. The TR archives of RIACA are incomplete. Earlier versions of this paper can be found at the "old OpenMath Home Pages" archived at the Uni. Köln.

[2] Henk Barendregt. Towards an interactive mathematical proof mode. In Fairouz Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, volume 28 of *Kluwer Applied Logic Series*, pages 25–36. Kluwer Academic Publishers, November 2003.

[3] Georg Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (part 1). *Mathematische Annalen*, 46:481–512, 1895.

[4] Georg Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (part 2). *Mathematische Annalen*, 49:207–246, 1897.

[5] Augustin-Louis Cauchy. *Cours d'Analyse de l'École Royale Polytechnique*. Debure, Paris, 1821. Also in *Œuvres Complètes* (2), volume III, Gauthier-Villars, Paris, 1897.

[6] W3C (World Wide Web Consortium). Mathematical markup language (MathML) version 2.0. W3C Recommendation, October 2003.

[7] W3C (World Wide Web Consortium). RDF Primer. W3C Recommendation, February 2004.

[8] W3C (World Wide Web Consortium). XQuery 1.0 and XPath 2.0 data model (XDM). W3C Recommendation, 2007.

[9] Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Inform. & Comput.*, 76(2/3):95–120, 1988.

[10] N.G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In *Workshop on Programming Logic*, 1987. Reprinted in [32, F.3].

[11] Richard Dedekind. *Stetigkeit und irrationale Zahlen*. Vieweg & Sohn, Braunschweig, 1872. Fourth edition published in 1912.

[12] Gottlob Frege. *Begriffsschrift: eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. Can be found on pp. 1–82 in [38].

[13] Gottlob Frege. *Grundgesetze der Arithmetik*, volume 1. Hermann Pohle, Jena, 1893. Republished 1962 (Olms, Hildesheim).

[14] Gottlob Frege. *Grundgesetze der Arithmetik*, volume 2. Hermann Pohle, Jena, 1903. Republished 1962 (Olms, Hildesheim).

[15] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. W. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer, 1980.

[16] Thomas L. Heath. *The 13 Books of Euclid's Elements*. Dover, 1956. In 3 volumes. Sir Thomas Heath originally published this in 1908.

[17] Fairouz Kamareddine, Twan Laan, and Rob Nederpelt. *A Modern Perspective on Type Theory from Its Origins Until Today*, volume 29 of *Kluwer Applied Logic Series*. Kluwer Academic Publishers, May 2004.

[18] Fairouz Kamareddine, Robert Lamar, Manuel Maarek, and J. B. Wells. Restoring natural language as a computerised mathematics input method. In MKM '07 [31], pages 280–295.

[19] Fairouz Kamareddine, Manuel Maarek, Krzysztof Retel, and J. B. Wells. Digitised mathematics: Computerisation vs. formalisation. In *Review of the National Center for Digitization*, volume 10, pages 1–8, Faculty of Mathematics, Belgrade, Serbia, 2007.

[20] Fairouz Kamareddine, Manuel Maarek, Krzysztof Retel, and J. B. Wells. Gradual computerisation/formalisation of mathematical texts into Mizar. In Roman Matuszewski and Anna Zalewska, editors, *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar and Rhetoric*, pages 95–120. University of Białystok, 2007. Under the auspices of the Polish Association for Logic and Philosophy of Science.

[21] Fairouz Kamareddine, Manuel Maarek, Krzysztof Retel, and J. B. Wells. Narrative structure of mathematical texts. In MKM '07 [31], pages 296–311.

[22] Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Flexible encoding of mathematics on the computer. In *Mathematical Knowledge Management, 3rd Int'l Conf., Proceedings*, volume 3119 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 2004.

[23] Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Mathlang: Experience-driven development of a new mathematical language. In *Proc. [MKMNET] Mathematical Knowledge Management Symposium*, volume 93 of *ENTCS*, pages 138–160, Edinburgh, UK (2003-11-25/---29), February 2004. Elsevier Science.

[24] Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Toward an object-oriented structure for mathematical text. In *Mathematical Knowledge Management, 4th Int'l Conf., Proceedings*, volume 3863 of *Lecture Notes in Artificial Intelligence*, pages 217–233. Springer, 2006.

[25] Fairouz Kamareddine and Rob Nederpelt. A refinement of de Bruijn's formal language of mathematics. *J. Logic Lang. Inform.*, 13(3):287–340, 2004.

[26] Toshihiro Kanahori, Alan Sexton, Volker Sorge, and Masakazu Suzuki. Capturing abstract matrices from paper. In *Mathematical Knowledge Management, 5th Int'l Conf., Proceedings*, volume 4108 of *Lecture Notes in Computer Science*, pages 124–138. Springer Berlin / Heidelberg, 2006.

[27] Michael Kohlhase. *An Open Markup Format for Mathematical Documents, OMDoc (Version 1.2)*, volume 4180 of *Lecture Notes in Artificial Intelligence*. Springer, 2006.

[28] Edmund Landau. *Grundlagen der Analysis*. Chelsea, 1930.

[29] Edmund Landau. *Foundations of Analysis*. Chelsea, 1951. Translation of [28] by F. Steinhardt.

[30] Manuel Maarek. *Mathematical Documents Faithfully Computerised: the Grammatical and Text & Symbol Aspects of the MathLang Framework*. PhD thesis, Heriot-Watt University, Edinburgh, Scotland, June 2007.

[31] *Mathematical Knowledge Management, 6th Int'l Conf., Proceedings*, volume 4573 of *Lecture Notes in Artificial Intelligence*. Springer Berlin / Heidelberg, 2007.

[32] Rob Nederpelt, J. H. Geuvers, and Roel C. de Vrijer. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1994.

[33] Giuseppe Peano. *Árithmetices Principia, Nova Methodo Exposita*. Bocca, Turin, 1889. An English translation can be found on pp. 83–97 in [38].

[34] P. Rudnicki. An overview of the Mizar project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, 1992.

[35] Alan Sexton and Volker Sorge. The ellipsis in mathematical documents. Talk overhead images presented at the IMA (Institute for Mathematics and its Applications, University of Minnesota) "Hot Topic" Workshop The Evolution of Mathematical Communication in the Age of Digital Libraries held on 2006-12-08/---09.

[36] Lambert S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH System*. PhD thesis, Eindhoven, 1977. Partially reprinted in [32, B.5,D.2,D.3,D.5,E.2].

[37] Joris van der Hoeven. GNU TeXmacs. *SIGSAM Bulletin*, 38(1):24–25, 2004.

[38] J. van Heijenoort. *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Harvard University Press, 1967.

[39] Alfred North Whitehead and Bertrand Russel. *Principia Mathematica*. Cambridge University Press, 1910–1913. In three volumes published from 1910 through 1913. Second edition published from 1925 through 1927. Abridged edition published in 1962.

[40] F. Wiedijk. Formal proof sketches. In *Proceedings of TYPES'03*, volume 3085 of *LNCS*, pages 378–393. Springer-Verlag, December 2004.

[41] Ernst Zermelo. Untersuchungen über die Grundlagen der Mengenlehre (part 1). *Mathematische Annalen*, 65:261–281, 1908. An English translation can be found on pp. 199–215 in [38].