Flexible Encoding of Mathematics on the Computer

Fairouz Kamareddine, Manuel Maarek, and J. B. Wells

Heriot-Watt University, http://www.macs.hw.ac.uk/ultra/

Abstract. This paper reports on refinements and extensions to the MathLang framework that add substantial support for natural language text. We show how the extended framework supports multiple views of mathematical texts, including natural language views using the exact text that the mathematician wants to use. Thus, MathLang now supports the ability to capture the essential mathematical structure of mathematics written using natural language text. We show examples of how arbitrary mathematical text can be encoded in MathLang without needing to change any of the words or symbols of the texts or their order. In particular, we show the encoding of a theorem and its proof that has been used by Wiedijk for comparing many theorem prover representations of mathematics, namely the irrationality of $\sqrt{2}$ (originally due to Pythagoras). We encode a 1960 version by Hardy and Wright, and a more recent version by Barendregt.

1 On the way to a mathematical vernacular for computers

Mathematicians now use computer software for a variety of tasks: typing mathematical texts, performing calculation, analyzing theories, verifying proofs. Software tools like Mathematica have been refined through many years of development. Research in mathematics, logic, and computer science has led to computer algebra systems (CAS's) and theorem provers (TPs). Languages like OMDoc [21] show good promise of having a universal way to share CAS and TP data in a mathematical software network [22].

Nevertheless, ordinary mathematicians still *write* mathematical knowledge (MK) using the traditional *common mathematical language* (CML) and typesetting tools like LaTeX. CML is mature and capable for human communication of mathematics, but unfortunately is difficult to computerize in a way that captures its essential structure.

We believe this is largely because existing computer MK representations either fail to capture the mathematical structure of natural language as used in mathematics or require writing in a rigid format [7]. Mathematical typesetting systems like LaTeX fail to capture the mathematical structure of both natural language text and symbolic formulas. Existing computer mathematics systems that capture mathematical structure either (1) do so for symbolic formulas but not natural language sentences and phrases (e.g., OpenMath or computer algebra systems), (2) handle natural language text via very complicated type systems (e.g., GF [17]), (3) require full formalization (e.g., Mizar [18] or other proof systems), or (4) otherwise restrict mathematicians' freedom to edit and display their texts in the form that best meets their needs and desires.

Theoretical approaches to this issue include De Bruijn's Mathematical Vernacular (MV) [2] and the recently developed Weak Type Theory (WTT) of Kamareddine and

Nederpelt [12], both of which strive toward a formalism that captures the structure of mathematical text while remaining close to the mathematician's original text.

Carrying the work of MV and WTT forward, we have been building MathLang, a framework for encoding mathematical texts on the computer. MathLang's heart is a formalism capturing the important mathematical structure of mathematical writing while keeping the flexibility of the traditional common mathematical language (CML). MathLang aims to interface computer mathematics systems with mathematicians.

MathLang analyses all mathematical texts into two interleaved parts, one for the natural language and one for the symbolic structure. The natural language part represents the text as the mathematician expects to view it. The symbolic part is automatically checked for structural correctness and contains enough semantic information to support further automatic manipulation by other available computer mathematics tools.

MathLang's design is constrained by the desire to balance the needs of ordinary mathematicians in writing MK with the needs of automated computer manipulation of MK. To support mathematicians, MathLang seeks to be (1) *expressive*, so it can handle all kinds of mathematical mental constructs, (2) *flexible*, so that ordinary mathematicians will not find it awkward to use, and (3) *universal*, so that it covers many branches of mathematics and is adaptable to new ones. To support computer manipulation, MathLang seeks to be (4) relatively *unambiguous*, so that automated processing will often be possible without human interaction, (5) sensibly *organized*, so that most desired ways of browsing the data will not be difficult, and (6) *automation-friendly*, to facilitate further more complex computations.

The language formalism of MathLang has three main features. **(1) A symbolic structure.** MathLang encodes mathematical texts via a symbolic syntax. A small set of grammatical constructions allows encoding the reasoning structure of texts. The symbolic structure is helpful for further encodings and translations. **(2) A CML layer.** MathLang can coat the symbolic structure of the text with natural language information that supports a CML view of the text, like the kind of visual output that could be generated if it was written in LaTeX, but also keeping the full underlying computerized structure. **(3) Automatic checking of basic grammatical conditions of the reasoning structure.** MathLang encodes the reasoning structure of a texts. A set of typing rules checks basic grammatical conditions using weak types, thus validating the good formation of the text at a grammatical level. It is very important that this checking does not require full formalization or committing to any specific foundation of mathematics.

In earlier work [13], we introduced MathLang, defined its XML-based concrete syntax, implemented a weak type checker, and tested the encoding of substantial real mathematical documents. MathLang's CML layer is new and is reported here for the first time. MathLang's symbolic structure and automated checking have had minor improvements since first reported, mainly to better support the integration of natural language text.

This article introduces MathLang's CML support in the context of explaining how the design of MathLang is evolving so that it can balance the needs both for the encoding to easily support desired computations and for the representation to be close enough to the thinking of the mathematician. Section 2 presents the MathLang philosophy while the later sections explain the capabilities of MathLang via example encodings. Although

we are currently testing MathLang by encoding two large mathematical books [14, 10], this article presents encodings of shorter examples, fully developed in MathLang. Section 3 presents an example (translated earlier into WTT) and shows its encoding in MathLang via both the symbolic and CML views automatically derived by MathLang. Section 4 presents a bigger example: *Pythagoras' proof of the irrationality of* $\sqrt{2}$. We chose this proof because it has been previously used by Wiedijk as a *universal* example of proof encoding for theorem provers [19]. We encode two informal versions of this proof in MathLang: an earlier one due to Hardy and Wright as well as a more explicit (but still informal) one due to Barendregt. For all 3 example encodings presented in this paper, the CML view illustrates that (unlike WTT) MathLang indeed preserves all the original text, including the natural language part.

2 MathLang's philosophy and evolving design

MathLang follows these goals:

1. Remain close to CML as used by mathematicians to write mathematics.
2. Act as a formal structured style which can be plugged into CAS's, TPs and other mathematical software systems.
3. Provide users of mathematics with much needed help in getting the original mathematical text into the computer by providing both a formal view of the text, and a natural language view. The formal view of the text is exactly its symbolic part and is passed to an automatic weak type checker which tests the structural validity of the text. The natural language view is automatically generated from the formal view and looks exactly like the original text.

We believe MathLang is the first framework which satisfies all the goals above. Some systems, Mizar [18] being the best example, have impressive libraries that show that much mathematics can be computerized and formalized, but they are not yet widely used by mathematicians. Abstract languages like MV and WTT are strongly based on goal 1, but fall short on goals 2 and 3, because neither provides any computer help, nor an automatic type checker to check the structural correctness of texts, nor is there a method yet of taking the translation of an original text in MV or WTT and deriving from the translation a text that looks exactly like the original one. Although MathLang has a type system which is an extension of WTT with flags and blocks, it goes well beyond MV and WTT by also working as a computer system which will communicate with other computer systems via XML and XSLT and will offer the user software help. Moreover, the merge between the abstract and the software could not have worked well to provide a view that looks so much like the original text, without both (1) the careful separation of texts into a natural language part and a formal part and (2) a careful interleaving of both parts.

Below we describe the framework MathLang following its three main features listed in section 1. We first describe the syntax and grammar of MathLang, then we describe how the CML layer has been designed, then we explain the value that is added by MathLang's type system compared to a normal CML encoding.

2.1 The symbolic structure

The MathLang symbolic language [13] has a strict grammar that allows one to describe, with a small set of constructions, any feature that composes the reasoning content of a mathematical text.

MathLang's grammar. As in MV [2] and WTT [12], an entire MathLang document is called a *book*. It is composed by a set of blocks, flags and lines. Blocks and flags are themselves composed by sub-blocks, flags and lines. A *block* highlights a piece of text as a coherent entity. It could be a section, a proof or a subproof, an example or any structure that the author wants to identify. Local constants are defined within a block (see below for more explanation). A *flag* declares variables (again, see below) and makes assumptions that will hold on a piece of text. A *line* is an atomic step of reasoning. As in common mathematical texts, a line could either make a new statement in the theory or define a new symbol (the *sentence level* groups these two possible kinds of line). Books, blocks, flags and lines compose the *discourse level*.
Four constructions are defined in MathLang to write expressions that will be the material of assumptions, declarations, definitions and statements from the discourse level. A *variable instance* refers to an already declared variable. A *constant call* uses a defined constant by referring to its definition and, in the case of a parametrized constant, by instantiating it. *Binding* an expression and a new variable is also possible with one of the many binders of the language. The last construction *attributes* an adjective to a noun to refine its meaning (again, see in the following for explanation on adjectives and nouns). These four constructions make up the *phrase level*
Symbols used in MathLang texts are used in the *atomic level*. They could be of three kinds: *variables* that are an abstraction of a mathematical object, *constants* that are parameterizable shortcuts for mathematical objects, and *binders*.

Grammatical categories. Any construction of the phase level is part of a specific grammatical category depending on the sort of the mathematical object described. In an obvious sense it depends on the symbol used to construct the expression in question. There are five of them: *terms* for elements designing a mathematical object, *sets* for sets of objects, *nouns* for kinds of mathematical objects, *adjectives* for elements that gives some attributes to a noun, and *statements* for expression that are considered as structurally valid statements. This grammatical information will be used by the MathLang type system (see section 2.3).

Concrete syntax. The *concrete syntax* of MathLang uses XML. XML-MathLang texts have level-based structure and contain grammatical information for each symbol.

2.2 The CML layer

A MathLang author separates a CML-text into its natural language part and its symbolic part, saving the natural language part for coating purposes later, and translating the symbolic part into XML-MathLang. This XML-MathLang text is then passed into the automatic type checker of the symbolic framework to test its structural well-formedness (see section 2.3) and thus implicitly also the original text. Figure 1 informally compares this encoding approach to others.

CML The overall document is in an informal encoding.

OMDoc's approach A slightly formal structure (dashed triangle) covers the entire document. CML texts are spread all around the document. OpenMath objects are used for formal definitions (`<FMP>` tag). Informal definitions (`<CMP>` tag) are text with embedded OpenMath formulas.

WTT The overall document is a formal encoding. Like N.G. de Bruijn's MV and Z. Luo and P. Callaghan's Mathematical Vernacular [15], WTT is a theoretical language and does not have any natural language representation.

MathLang's approach A computerized structure covers the entire documents. Pieces of CML transformation procedures are attached to nodes of the document's skeleton.

TPs' approach The document is a fully formal data. Usually, CML explanations are separately given (dashed blobby shapes). The natural language is produced with generic computations according to some structured text given by the programmer. (We consider here the generation of natural language texts from Coq [3], the design of an electronic library of mathematics [4], the MoWGLI project [5], work to interface Coq with $\TeX_{\text{MACS}}$ [8], and the documentation system of FoC [16].)

Figure 2 shows that the original CML text is first divided into a symbolic part and a natural language part and that afterward, the full original text can be retrieved, as well as a fully symbolic part which can be passed for further processing. We are currently using XSLT to express the transformations to CML. We are considering the use of the Grammatical Framework [17] for its expressiveness as a replacement of XSL. A possible intermediate stage from CML-texts to fully formalized texts (the "later computations" in figure 2) could be to use the Mathematical Proof Language [1], a language between informal and formalized mathematics.

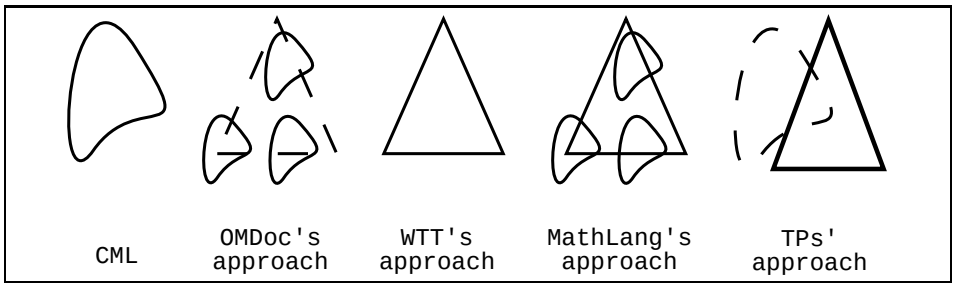


Fig. 1. Approaches. Informal data is represented by blobby shapes (⬭). Computerized and more formal data is represented by triangles (△).

Rendering tools have been developed using XSLT to generate representations of MathLang documents. All the examples of this article have been automatically generated using these tools which work as follows given an XML-MathLang text:

- Display the information in symbolic structural view.
- Display the information in CML form (if it contains a natural language coating).

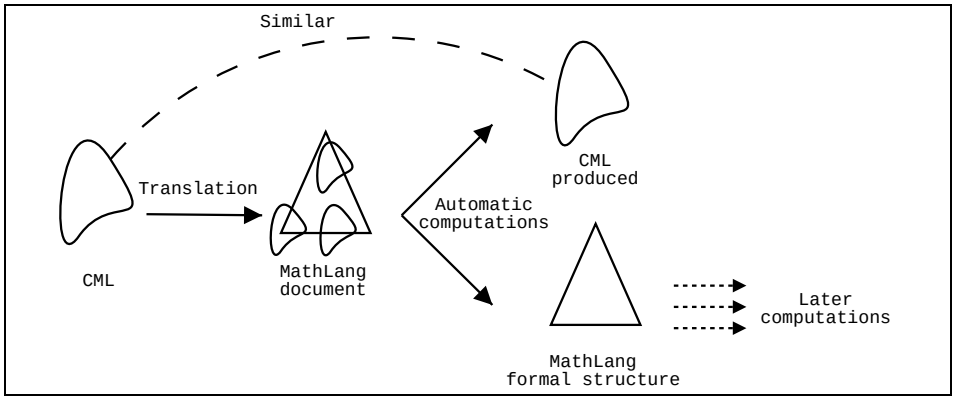


Fig. 2. Translation process

2.3 MathLang typing

From this grammatical information given in the writing down of MathLang document, computerized analysis is possible. It gives some properties about the coherence of the text. This is done by assigning weak types to each grammatical category and construction. This weak typing remains at a grammatical level. A type theory could then be used to check the text. If a book is considered as a valid book after typing, it means that variables are well declared before being used and constants are well used according to their definitions. Because this analysis remains at a grammatical level, it therefore does not deal with the logical truth or validity of the text.

We developed an *automatic weak type checker* which analyzes the coherence of the XML-MathLang text. This automatic type checker, when given a text in XML-MathLang, checks whether the reasoning structure of the text is valid or not. In the latter case, an error message gives the reason of failure.

3 A working example

The translation of a mathematical text into a MathLang document is done by a human assisted by a computer. The MathLang writer should have a mathematical background to follow the reasoning of the mathematician author of the original text. Moreover, the MathLang writer needs to have skills in computer science to encode the document into the MathLang XML syntax. This last requirement would be avoided in the future by the development of a dedicated editor for MathLang (see section 5).

3.1 The translation process

Four actions compose the translation process of a CML text into MathLang with presentation directives. The first two are to be done by the MathLang user. They are represented by the `Translation` arrow of figure 2. The last two are initiated by the user

and are automatically executed by MathLang's framework. These are the `Automatic computations` of figure 2.

Translating. This is the main work done by the user: it reveals the reasoning structure of the original text and encodes it into MathLang. These do not require more work than a normal study of the text. The writer just needs to unfold implicit information form his understanding of the CML text (see section 3.2).

Adding natural language. This is a crucial stage of the translation. It will not influence the *deep* MathLang encoding of the text and so will not change its validity. But it will make the MathLang document as readable as the original CML text. Because this stage is independent from type checking, this information could be added during the early translation of a piece of text or after the validating stage.

Type checking. This stage is an automatic computation. The MathLang framework has a built in type checker which takes a MathLang XML document and checks its structural validity. The type checking of a MathLang XML document returns true if the program goes through the document without finding any problem. Otherwise the MathLang type checker will point an element of the document where an error has been found. The user will then need to fix it and recall the checker again.

Producing the CML output. This stage is again fully automatic. It uses several XSL transformations to generate a CML text with colour annotations showing the weak types in the document. This process takes into account the presentation information given by the author to generate an output which is close to the original CML text.

To illustrate the steps to be taken by a MathLang user, we will explain the two first stages with a concrete example. We give the original text to be encoded, a representation of our MathLang encoding, and the CML output obtained from it. We use grey scale to show the belonging of text parts to a grammatical category. This helps to show in the CML MathLang output, the underlying structure of the MathLang encoding. Figure 4 contains the grey scale coding we are using in this paper.

We took our example from article [12]. By using the same example we aim to show what improvements MathLang makes over WTT. WTT was a theory describing a type system for the first level of formalization of mathematical texts. The language description was not precise enough to be used and no implementation of its ideas were available. MathLang reuses this type theory in its implementation. New constructions have been added as explained in [13].

3.2 The example

Our example is a definition which defines the *difference quotient* of a function, explains what is a *differentiable* function and then states that $\sqrt{|x|}$ is not differentiable at 0. The original CML text is given by figure 3. Figure 5 shows our MathLang encoding and figure 6 the output we get.

Translating. The original text is titled "Definition" and contains three steps. We encode it by a block (with indices $\{1\}$) to reflect the grouping of the overall definition. Then to each step corresponds one line. The first one (numbered 1) stands for the definition of the difference quotient and will be a definition line in MathLang. The second (numbered 2) will again be a definition line defining a new notion: differentiable. The

Definition 1. *Let $h \neq 0$, let f be a function from A to $\mathbb{R}$, $a \in A$ and $a + h \in A$. Then $\frac{f(a+h)-f(a)}{h}$ is the* difference quotient *of f in a with difference h. We call f* differentiable *at a if $\lim_{h\to 0} \frac{f(a+h)-f(a)}{h}$ exists. The function $\sqrt{|x|}$ is not differentiable at 0.*

Fig. 3. *Difference quotient* example: original text

T Terms | S Sets | N Nouns | A Adjectives | P Statements | Z Declarations
Γ Contexts | L Lines | F Flags | K Blocks | B Books

Fig. 4. Grey scale coding

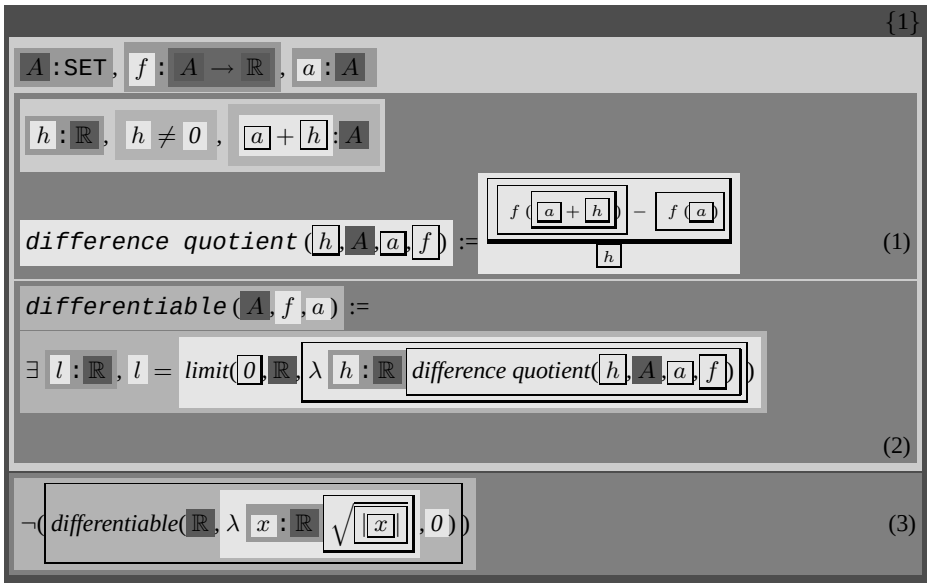


Fig. 5. *Difference quotient* example: symbolic structural view of MathLang

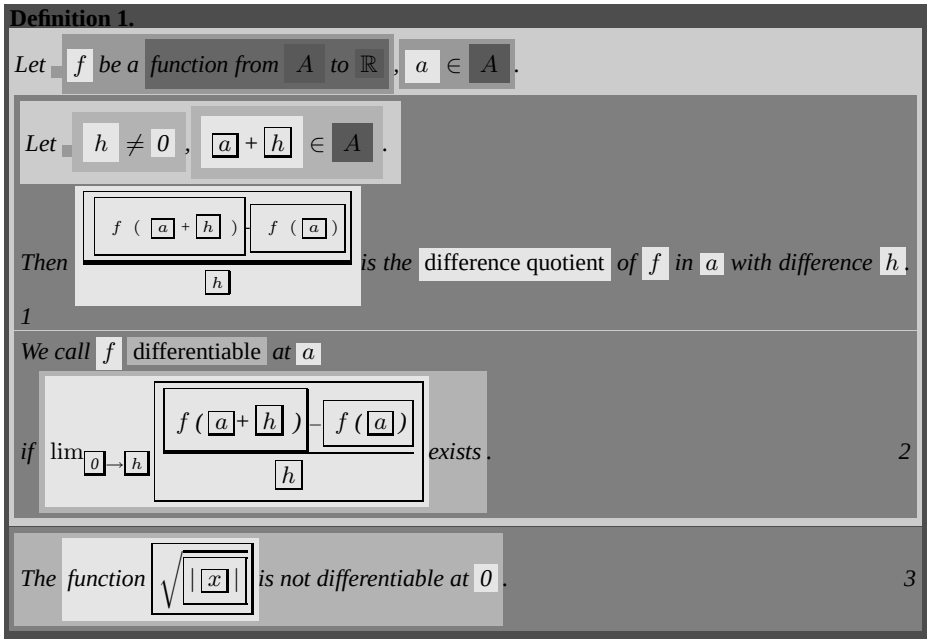


Fig. 6. *Difference quotient* example: CML view of MathLang

last step (numbered 3) that establishes that $\sqrt{|x|}$ is not differentiable at 0 is encoded by a statement line.

The definitions. In the text, we notice that both definitions have some elements in common. They use the same set A, the same function f and the same element a of A. These are variables. To put them in common in the MathLang document we use the flag construction that introduces context elements for several lines. In our case three variables are declared for the first two lines. The first definition also requires the declaration of the variable h together with two assumptions: $h \neq 0$ and $a + h$ is in A.

We notice that the flag contains three elements. In the original text only four context elements are visible (the first sentence of figure 3). The MathLang grammar forces one to strictly declare any variable. $A : SET$ that declares a set variable A is an example of implicit declaration which is revealed by MathLang.

With these contexts enunciated, we give our two definitions. Here again some information will become explicit. Constant parameters should correspond to the current set of declared variables: h, A, a and f for the difference quotient and A, f and a for differentiable notion. The first definition uses some constants that we consider to be defined earlier on. The second definition uses the existential binder to state that a limit exists and that its value is the limit of the differential quotient when h tends towards 0. Our choice here was to reuse the constant previously defined `difference quotient` instead of writing again the equation.

The statement. The last line with an empty context states that the function $\sqrt{|x|}$ (encoded with the λ binder) is not differentiable at 0. This expression uses the newly defined constant `differentiable`.

Adding natural language. The block including the entire text is represented with a definition LaTeX environment. The sentence "Let 1, 2," stands for the flag of our encoding where 1 and 2 are the declarations of f and a. An empty little box in figure 6 shows that an implicit declaration (the set A) took place. We do the same for the context of line 1. The order of declarations and assumptions was changed from figure 3 to figure 6 on purpose to show the boxes that correspond to the flag and to the context of the first line. "Then 1 is the 2 of 3 in 4 with difference 5" is the template for our first definition. 1 being the expression of the definition, 2 the constant's name, 3 the constant's fourth parameter, 4 the third one and 5 the first one.

These associations that link one construction of the language to a natural language template could either be defined globally or locally to the document. One association could be reused at different positions. They are defined using XSL with some MathLang specific commands to easily refer to the information contained in the encoding. For example a unique command is used to colour the pieces of text according to their grammatical categories. For example, the presentation information for the line numbered 1 is as follow.

```
<template output="cml.tex" kind="xsl">
  <categ kind="par" boxed="no">
    <xsl:apply-templates select="context"/>
    <xsl:text>Then </xsl:text>
    <xsl:apply-templates select="expression"/>
    <xsl:text> is the </xsl:text>
    <xsl:apply-templates select="constant"/>
    <xsl:text> of </xsl:text>
    <xsl:apply-templates select="parameters[4]"/>
    <xsl:text> in </xsl:text>
    <xsl:apply-templates select="parameters[3]"/>
    <xsl:text> with difference </xsl:text>
    <xsl:apply-templates select="parameters[1]"/>
    <number/>
  </categ>
</template>
```

This information is a mix of XSL standard elements and MathLang specific ones. A first process will transform all these presentation data that coat the document to produce one single XSL file specific to the document in question. The second process simply consists of applying these transformation to the document itself.

4 Pythagoras' proof of irrationality of $\sqrt{2}$

In this section we give the CML-MathLang view of two versions of *Pythagoras' proof of the irrationality of* $\sqrt{2}$. We chose this proof because it has been previously used by Wiedijk as a *universal* example of proof encoding for theorem provers [19, 20]. Both original versions are included in [19]. The first one is an informal version written by G. H. Hardy and E. M. Wright (see figure 7). Section 4.1 is the CML view of our translation into MathLang. The second one is a more explicit proof written by Henk Barendregt. We give the CML view of our MathLang translation in section 4.2.

Theorem 1 (Pythagoras' Theorem). $\sqrt{2}$ *is irrational.*

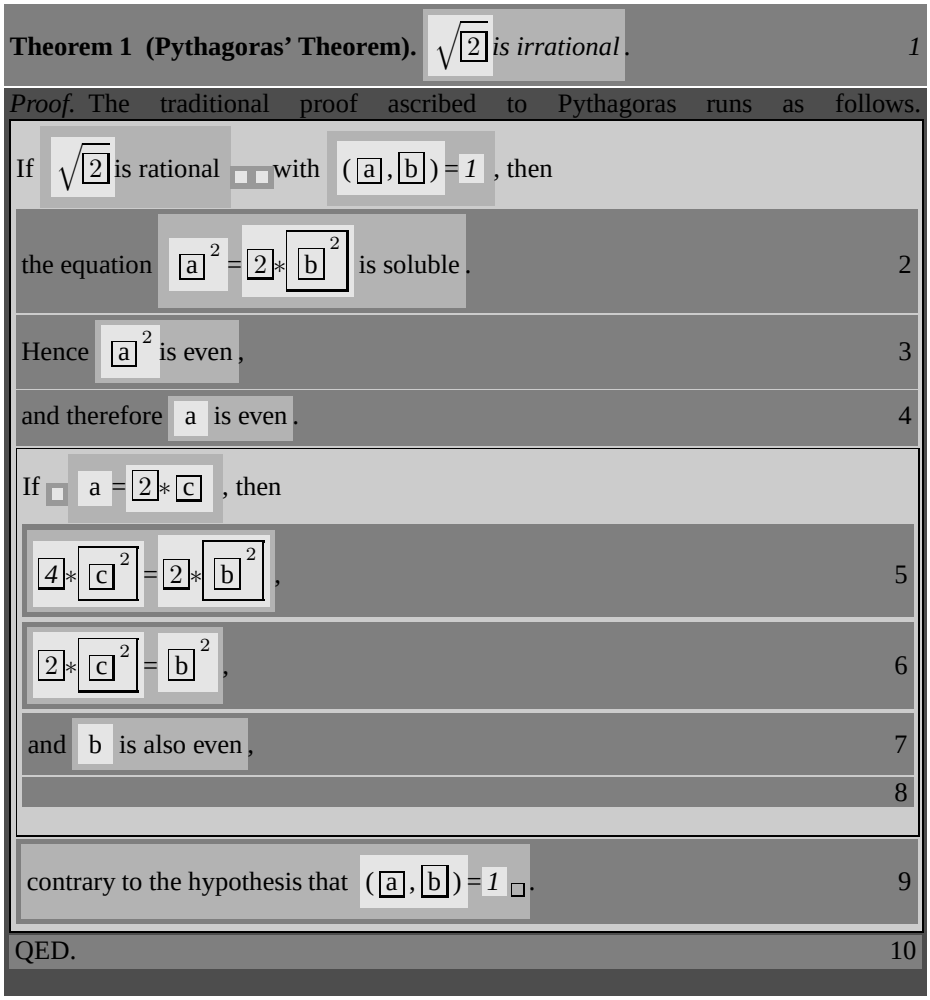
Proof. If $\sqrt{2}$ is rational, then the equation

$$a^2 = 2b^2$$

is soluble in integers a, b with $(a, b) = 1$. Hence a^2 is even, and therefore a is even. If $a = 2c$, then $4c^2 = 2b^2$, $2c^2 = b^2$, and b is also even, contrary to the hypothesis that $(a, b) = 1$.

Fig. 7. Proof of the irrationality of $\sqrt{2}$

4.1 The informal proof of Hardy and Wright



4.2 A more explicit informal proof by Barendregt

Lemma 1. *For* m, $n \in \mathbb{N}$ *one has* $m^2 = 2n^2 \Rightarrow m = n = 0$. *1*

Proof. 1.1

Define on $\mathbb{N}$ the predicate $P(m) \Leftrightarrow \exists n.\ m^2 = n^2 \,\&\, m > 0$. 2

Claim: $P(m) \Rightarrow \exists m' < m.\ P(m')$. 3

Indeed suppose $m^2 = 2n^2$ and $m > 0$. It follows that m^2 is even, 4

but then m must be even, as odds square to odds. 5

So $m = 2k$ 6 and we have $2n^2 = m^2 = 4k^2$ 7

$\Rightarrow n^2 = 2k^2$ 8

Since $m > 0$, it follows that $m^2 > 0$, 9 $n^2 > 0$ 10

and $n > 0$. 11 Therefore $P(n)$. 12 Moreover $m^2 = n^2 + n^2$ 13

$> n^2$ 14 so $m^2 > n^2$ 15 and hence $m > n$. 16

So we can take $m' = n$. 17

By the claim $\forall m \in \mathbb{N}.\ \neg P(m)$ since

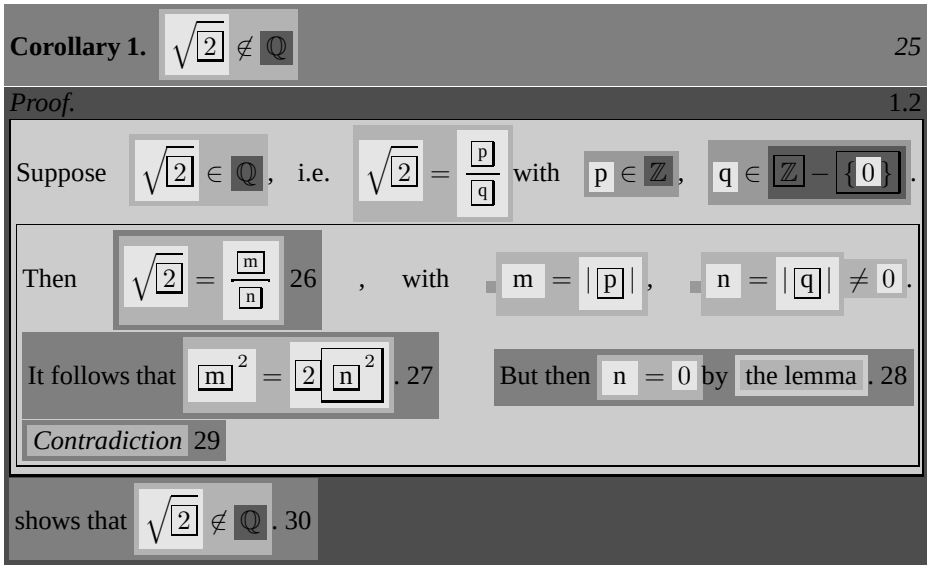
there are no infinite descending sequences of natural numbers. 18

Now suppose $m^2 = 2n^2$ with $m \neq 0$. Then $m > 0$ 19

and hence $P(m)$. 20 *Contradiction*. 21 Therefore $m = 0$. 22

But then also $n = 0$. 23

24



4.3 Discussions

More or less explicit texts. The small set of grammatical constructions of MathLang gives all the material needed to express reasoning of different levels of precision. Both versions given here follow the same kind of structures in their demonstrations. Small steps leading to bigger ones. New statements in a certain context and definitions of new symbols. This is what MathLang is designed for: encoding these simple demonstration constructions into a structured document that eases computation. In both cases the MathLang encoding follows the reasoning structure of the text. Simple transformation procedures are then given to get the original text in return. These two versions of the same proof show the expressiveness of MathLang: we have used the same language to write a non precise proof as well as a fully explicit one. From this encoding of the proof in MathLang we get both the original text and the computerized document.

Moving from MathLang to other encodings. As we said before, the symbolic structure of MathLang texts eases both the automatic computations on the text and further translations to other languages. To illustrate this we have translated Barendregt's version of the proof into OMDoc/OpenMath.

The translation process is the same as the one carried out to obtain the CML view of MathLang documents. We spread the document with transformation information. This led to a big program that transforms automatically the MathLang document into OMDoc/OpenMath. In this transformation `<CMP>` (informal) and `<FMP>` (formal) OMDoc's tags could easily be informed. The first one using the same process as the one carried out to obtain a CML view of the MathLang text. The second one by using the symbolic structure to obtain OpenMath formulas. The third part of this translation consists in mapping the MathLang basic structures into OMDoc constructions.

For example, our MathLang translation of Barendregt's version of the proof consists of one line followed by one block, one line and one block. The first line being the definition of the lemma, the first block its proof, the second line (numbered 25) is the definition of the corollary and the second block its proof. In OMDoc the lines 1 and 25 are `<assertions>` and the blocks are `<proof>`. The predicate P defined in the proof of the lemma (line 2) is a symbol definition (`<symbol>`) in the overall proof (`<theory>` in OMDoc, `book` in MathLang).

We compared this translation, that leads from a CML text to an OMDoc document via a MathLang document, to a direct translation from CML to OMDoc. The direct translation seems to be quicker but the one via MathLang has three advantages.

- The MathLang document is checked by the MathLang weak type checking. This validates the good formation of the structure of the document. Such an analysis does not exist for OMDoc and OpenMath.
- OMDoc has a formal (`<FMP>`) and an informal (`CMP`) tag for data. There are no requirements in OMDoc to have them both informed. The computerisable content of an OMDoc document depends on the author. In MathLang the main skeleton of the document fully uses symbols (similar to OMDoc's formal data), the natural language is added on top of this structured content. A MathLang-text always provides a *formal* content that could be forgotten in OMDoc.
- This structured content is encoded using a small set of MathLang constructions. These simple grammatical constructions guide the author in the translation process. It is then easier with this guiding process to obtain well structured OMDoc formal data by translating first into MathLang than directly into OMDoc.

5 Future works

We described in this paper how we encode mathematics using MathLang and how we produce a CML view of this encoding. Our main future work is to ease the input of data inside the MathLang framework. Currently one requires skills in computer science to write a MathLang document. We are currently developing a user interface for MathLang based on the scientific editor TEX_{MACS}.

As we said earlier in this paper we are starting and planing to extend the current language and framework. We aim to do it by making a use of the Grammatical Framework of A. Ranta and by moving to a second level of MathLang encoding which will include more logic and semantic.

6 Conclusion

By providing at the same time a computable encoding and a simple structure to produce readable output, our language MathLang tries to fill the gap between printed mathematical texts and mathematical softwares. The main problem mathematicians face when using formal systems is that it is very difficult, even for an expert, to find her way in a formal proof written by someone else. As described in this article, our system provides a direct correspondence between a symbolic structure and a CML view of a text. The

original mathematical document is wisely partitioned between a natural language part and a symbolic part. The symbolic part can be used in more formal computations on the original text. This is how we imagine what could be the *new mathematical vernacular* for computers.

References

1. Barendregt, H.: Towards an Interactive Mathematical Proof Mode. In *Thirty Five Years of Automating Mathematics*, Kamareddine, Editor, Kluwer Applied Logic **28**, (2003).
2. de Bruijn, N.G.: The Mathematical Vernacular, a language for mathematics with typed sets. Workshop on Programming Logic (1987).
3. Coscoy, Y.: A Natural Language Explanation for Formal Proofs. LACL (1996).
4. Asperti, A., Padovani, L., Sacerdoti Coen, C., Guidi, F., Schena, I.: Mathematical Knowledge Management in HELM. AMAI **38(1-3)**, 27–46, (2003).
5. Mathematics On the Web: Get it by Logic and Interfaces (MOWGLI). `http://www.mowgli.cs.unibo.it/`
6. Davenport, J.H.: MKM from Book to Computer: A Case Study. LNCS **2594**, 17–29 (2003).
7. Théry, L.: Formal Proof Authoring: an Experiment. UITP (2003).
8. Audebaud, P., Rideau, L.: $\TeX_{\text{MACS}}$ as Authoring Tool for Publication and Dissemination of Formal Developments. UITP (2003).
9. Deach, S.: Extensible Stylesheet Language (XSL) Recommendation. World Wide Web Consortium (1999), `http://www.w3.org/TR/xslt`.
10. Heath: The 13 Books of Euclid's Elements. Dover (1956).
11. Heijenoort (van), ed.: From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931, Harvard University Press (1967).
12. Kamareddine, F., Nederpelt, R.: A refinement of de Bruijn's formal language of mathematics. Journal of Logic, Language and Information **13(3)**, 287–340, (2004).
13. Kamareddine, F., Maarek, M., Wells, J.B.: MathLang: Experience-driven Development of a New Mathematical Language. ENTCS **93**, 138–160, (2004).
14. Landau, E.: Foundations of Analysis. Chelsea (1951).
15. Luo, Z., Callaghan, P.: Mathematical vernacular and conceptual well-formedness in mathematical language. LNCS/LNAI **1582** (1999).
16. Maarek, M., Prevosto, V.: FoCDoc: The documentation system of FoC. Calculemus (2003).
17. Ranta., A.: Grammatical Framework: A Type-Theoretical Grammar Formalism. Journal of Functional Programming (2003).
18. Rudnicki, P., Trybulec, A.: On Equivalents of Well-foundedness. Journal of Automated Reasoning **23**, 197–234, (1999). Contains a general introduction to Mizar.
19. Wiedijk, F.: The Fifteen Provers of the World. University of Nijmegen.
20. Wiedijk, F.: Comparing Mathematical Provers. LNCS **2594**, 188–202, (2003).
21. Kohlhase, M.: OMDoc: An Open Markup Format for Mathematical Documents (Version 1.1). Technical report (2003).
22. Zimmer, J., Kohlhase, M.: System Description: The MathWeb Software Bus for Distributed Mathematical Reasoning. LNCS **2392** (2002).