Gradual Computerisation/Formalisation of Mathematical Texts into Mizar

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

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

Abstract. We explain in this paper the gradual computerisation process of an ordinary mathematical text into more formal versions ending with a fully formalised Mizar text. The process is part of the MathLang–Mizar project and is divided into a number of steps (called aspects). The first three aspects (CGa, TSa and DRa) are the same for any MathLang–TP project where TP is any proof checker (e.g., Mizar, Coq, Isabelle, etc). These first three aspects are theoretically formalised and implemented and provide the mathematician and/or TP user with useful tools/automation. Using TSa, the mathematician edits his mathematical text just as he would use LaTeX, but at the same time he sees the mathematical text as it appears on his paper. TSa also gives the mathematician easy editing facilities to help assign to parts of the text, grammatical and mathematical roles and to relate different parts through a number of mathematical, rethorical and structural relations. MathLang would then automatically produce CGa and DRa versions of the text, checks its grammatical correctness and produce a dependency graph between the parts of the text. At this stage, work of the first three aspects is complete and the computerised versions of the text, as well as the dependency graph are ready to be processed further. In the MathLang–Mizar project, we create from the dependency graph, the roles of the nodes of the graph, and the (preamble) of the CGa encoding, a Mizar Formal Proof Sketch (FPS) skeleton. The stage at which the text is transformed into a Mizar FPS skeleton has only been explained through transformation hints, and is yet to be theoretically developed into an aspect that can be implemented and developed into a partially-automated tool. Finally, the Mizar FPS skeleton of the text is transformed (currently by hand as any Mizar expert would do and without any computerised tools) into a correct Mizar FPS and then into a fully formalised Mizar version of the text. Although we have tested our process on a number of examples, we chose to illustrate it in this paper using Barendregt's version of the proof of Pythagoras' theorem. We show how this example text is transformed into its fully formalised Mizar version by passing through the first three computerised aspects and the transformation hints to obtain Mizar FPS version. This version is then developed by hand into a fully formalised Mizar version.

ISBN 978-83-7431-128-1 ISSN 0860–150X

1 Introduction

The past forty years have seen a growing number of uses of the computer in the daily routine of the mathematician. These uses range from authoring tools (e.g., LaTeX, MathML), to computation and calculation aids (e.g., Mathematica) to proof checking tools (e.g., Mizar). Proof checking tools have had the least uses by ordinary mathematicians since they are completely different from traditional mathematical authoring, and remain difficult to use by non experts. Even if the language behind the proof checking tool closely mimics the common mathematical language (CML – the language and style mathematicians use to write their mathematics), the formalisation process remains very long, labor-intensive and will require expertise in at least programming and logic. Furthermore, for a mathematical text to be fully verified by a proof checking tool, all its informal parts and proofs need to be rewritten in sufficient details before being processed for correctness. Mathematicians do not like writing proofs or details that they consider to be obvious or trivial. Furthermore, mathematicians prefer developing new or studying existing mathematical theories rather than proof checking using the computer existing theories. And so, the gap between the mathematician and the computer proof checker remains large.

Recent years have seen many attempts to bridge this gap. For example, some work has been done on computerising mathematical texts without fully formalising or computer proof checking them. Such computerisations are not sufficiently detailed for correctness verification but are used as *skeletons* in the full formalisation (see Wiedijk's work [25]). Although the computerised text remains at a low level to be fully automatically checked, it has a precise notion of correctness: it is *syntactically* correct according to the grammar language but according to the proof language it contains steps that are not sufficiently justified. However, in order to create these skeletons, the user still needs to be an expert in the final destination language. For example, for a mathematician to carry out the work as outlined in [25], he/she needs to be an expert in Mizar.

In this paper, we allow the ordinary mathematician to do a reasonable amount of work on computerising mathematical texts that lead to computerised versions that can be passed to any expert in any proof checker to be fully formalised in that proof checker. We choose Mizar to be our target proof checker, and we consider G. H. Hardy and E. M. Wright's skeletons as one of the steps to reach the final Mizar version. However, the skeleton and the Mizar version are obtained not from CML texts, but from versions computerised by the mathematician which have gone through a number of automatic checking and manipulations. These computerised versions are easier to transform into skeletons and final Mizar version.

Our approach is sketched in Figure 1. Instead of a Mizar expert transforming the CML text into a fully formalised Mizar version following one of the paths:

- ⓒ: immediately create the fully formalised Mizar version of the text;
- ⓑ-ⓔ: first create the Mizar Formal Proof Sketch skeleton and then the fully formalised Mizar version of the text,

we believe that each of the formalisation paths ⓒ and ⓑ-ⓔ could be divided into a number of smaller steps as in the path ⓐ-ⓓ-ⓔ of Figure 1 where all the levels

at step ⓐ are done by the mathematician and where the sub-path ⓓ-ⓔ is done by the Mizar expert. This approach has a number of advantages:

- It gives a better view at the process of computerisation.
- It helps build computer programs that can assist humans along the computerisation/formalisation processes. In fact, at the TSa, CGa and DRa levels, the user already enjoys numerous automated help which makes his work almost minimal. We also aim for partial automations of steps ⓓ and ⓔ.
- It shows that the mathematician can benefit by first authoring the text, and later on 'tagging' it within the MathLang system and checking its grammatical correctness (see Figure 2 and Section 2.1).
- As expressed in the description of Figure 1, different formalisation paths involve different levels of the expertise required by the user.

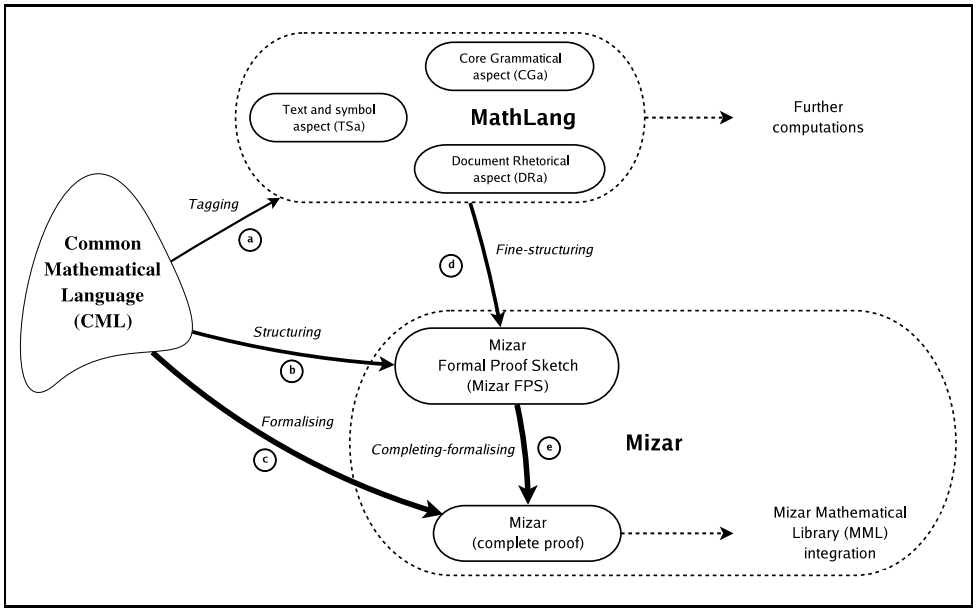


Fig. 1. Computerisation/formalisation paths from CML to Mizar.
The labeled arrows shows the computerising paths from CML to Mizar. In this paper we mainly focus on the path ⓐ-ⓓ-ⓔ. We also briefly compare it with the path ⓑ-ⓔ and the path ⓒ. The width of the arrow representing each path segment increases accordingly to the expertise required to achieve the path segment. The dashed arrows illustrate further computerisation that one can envision.

We have chosen Mizar as the final destination since it comes with the biggest library of mathematical knowledge verified by computer. This is important since, most CML texts assume that the reader has at least the fundamental mathematical knowledge required to follow the topic in the document. Therefore, if we want to formalize a mathematical text, we have to refer at some point to that fundamental knowledge. Mizar has an impressive library of mathematical knowledge: the Mizar Mathematical Library (MML). In addition to its MML, Mizar is the most accessible of the proof checkers (in terms of readability and write-ability) by mathematicians as expressed by the Mizar creator Trybulec:

> Experience has shown that many people with some mathematical training develop a good idea about the nature of the Mizar language just by browsing through a sample article. This is no big surprise, since one of the original goals of the project was to build an environment which supports the traditional ways that mathematicians work. *[20, pp.2]*

This said, the user will still need to have an expert knowledge if he/she wants to deal with the whole mechanism (i.e. the Mizar system, MML, the Mizar Language, the MML search engines) and create a new Mizar document.

1.1 Example

The example we use in this paper is taken from [26] where Wiedijk used Hardy and Wright's version [8, Ch. IV] of Pythagoras' theorem of irrationality of $\sqrt{2}$ to compare computer based theorem provers. Barendregt wrote a textual version [3] (reproduced in Figure 8, page 118) of this proof which is said to be *"informal"* in contrast to the formal versions of theorem provers as in [26]. Wiedijk's comparison illustrates the need to assist the mathematician non-expert in theorem provers. We have already used this example in [11] to obtain what was then a CGa version. In this paper, we use this example to show all the versions of the proposed path (TSa, CGa, DRa, Mizar FPS skeleton, Mizar FPS and finally Mizar).

1.2 Notations, contributions and outline

	MathLang	Mizar
Syntax	sans-serif	`typewriter`
Jargon	**boldface**	*slanted*

Notations. We adopt some rules to facilitate reading this paper. We use different font styles to differentiate MathLang and Mizar syntax and notions. MathLang's abstract syntax is written in the sans-serif font style, in contrast to Mizar's syntax, which is highlighted using the `typewriter` font style. MathLang jargon words are written using the **boldface** family font. Specific Mizar jargon words are written using the *slanted* font style.

Contributions. Our contributions can be summarised as follows:
1. *A gradual computerisation/formalisation into Mizar.* We propose a new approach for the gradual computerisation/formalisation of a CML document into its more formal versions ending with a fully formalised Mizar version. This gradual approach shows the increasing level of expertise required to achieve the fully formalised Mizar version, and allows the mathematician and the Mizar expert to collaborate in the process (see the explanation of Figure 1).
2. *Transformation hints.* Through the example, we give transformation hints that allows us to create a skeleton of a document in a fully formal Mizar language.
3. *A short comparison between MathLang and Mizar constructs.* Through the example, we compare different MathLang constructions with their counterparts in Mizar. This brief comparison provides hints and ideas on which knowledge is required to understand the original document, how this document could be stored in MathLang and in Mizar where identifiers are taken from MML.

Outline. In Section 2 we briefly explain the various boxes shown in Figure 1. We also explain the transformation path we are following to build a formal document in the Mizar language. In Section 3 we explain how the DRa annotation on the

CML text helps us to build a rough skeleton of the Mizar document. In Section 4 we give hints as to how mathematical identifiers and their CGa presentation could be used to narrow their representation in Mizar and give further transformation hints of the MathLang document into Mizar. In Section 5 we reflect on the different formalisation paths. Finally, in Section 6 we conclude and describe related and future work.

2 Background

2.1 MathLang and its aspects

Since 2001, the ULTRA group has been developing as part of the MathLang project, a number of prototypes for computerising mathematics. The project MathLang aims to give *alternative* and *complete* paths which transform mathematical texts into new computerised and/or formalised versions. These paths are intended to accommodate different degrees of formalisation, different mathematical editing/checking tools and different proof checkers. Dividing the formalisation of mathematical texts into a number of stages was first proposed by N.G. de Bruijn to relate CML to his Mathematical Vernacular [6] (MV) and his proof checking system Automath. We call this principle *de Bruijn's path*.

> The work may be subdivided. One can think of a first stage where a person with some mathematical training inserts a number of intermediate steps whenever he feels that further workers along the belt might have trouble, and a second stage where the logical inference rules are supplied and the actual coding is carried out. For the latter piece of work one might think of a person with just some elementary mathematics training, or of a computer provided with some artificial intelligence. But we should not be too optimistic about that: programming such jobs is by no means trivial.*[5]*

MV was proposed as a formal substitute for parts of CML. Nederpelt refined MV into another formal substitute for parts of CML, Weak Type Theory (WTT) whose underlying proof theory was developed by Kamareddine [17,14]. MathLang started from de Bruijn's path idea and Nederpelt's WTT and was faced with the huge challenge of how to really create a path from original mathematical texts into fully formalised ones and how would this path differ for different choices of texts, text editors, logical frameworks, and proof checkers. Soon after a number of prototypes were built, it became obvious that the stages of the path and the formal substitute of CML need to be seriously revised.

The MathLang language expressiveness has been increased and its description simplified in comparison with MV and WTT. Moreover, MathLang adopted to decompose the computerisation process by means of knowledge components. Each element of this decomposition is defined in what we call, an **aspect**. In the current development of MathLang we have defined three aspects (CGa, TSa and DRa) which we explain below. Figure 2 illustrates with a sentence from our example (see Section 1.1) the viewpoint each aspect gives to the same text.

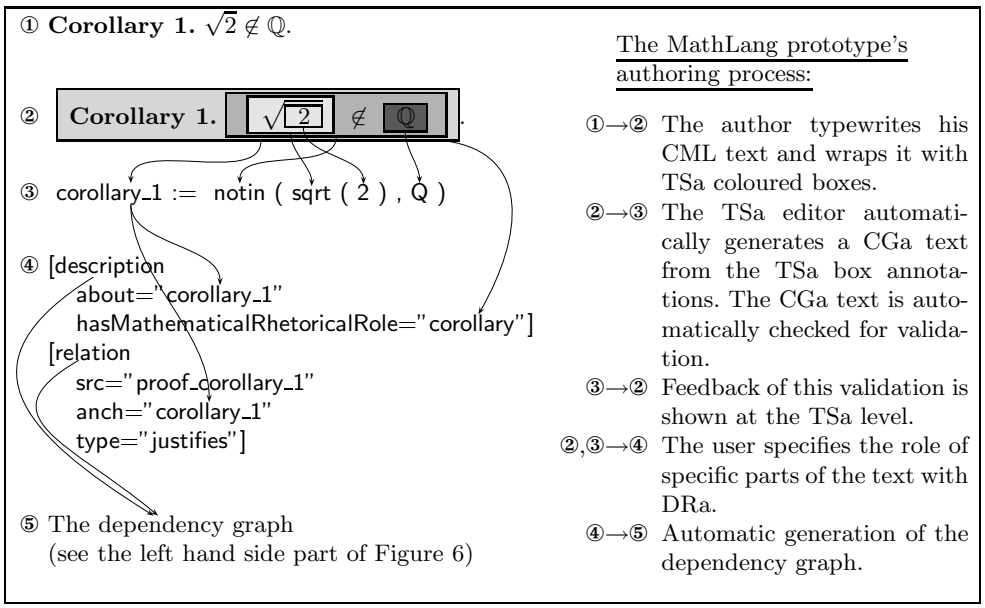


Fig. 2. Example of mathematical authoring using MathLang's aspects.

To illustrate the decomposition of mathematical knowledge with MathLang's aspects, we use the example in ①: the definition of Corollary 1 which states the irrationality of $\sqrt{2}$. We identify in this sentence the grammatical role of each element of the text: **definition** for the entire sentence, **term** for "$\sqrt{2}$" and "2", **set** for "$\mathbb{Q}$" and **statement** for "$\sqrt{2} \notin \mathbb{Q}$". The author attributes to each element its CGa grammatical role by wrapping it into a coloured box following our colour coding system of Figure 3. The formal interpretation of this sentence is automatically generated from TSa's ② and is printed in ③ using CGa's abstract syntax as defined in [13]. The identifiers corollary_1, notin, sqrt, 2 and Q are provided by the user as arguments for each coloured box of ②. Note that the CGa syntax used in ③ is not meant to be used by the end-user of MathLang, it is only designed for computerisation purposes. The MathLang end-user edits his MathLang document using the view offered by TSa, as shown by ② (TSa plays the role of a user interface for MathLang). The internal syntax used in our implementations follows XML recommendations. In ④ we indicate with DRa what role this sentence plays in the context from which it was taken. Sentence ① plays the role in the entire example (see [3] or Figure 8) of corollary and is being justified by the proof which follows. DRa's ④ makes this explicit (see DRa's role list in Table 1). Finally, we automatically produce a dependency graph ⑤ out of the DRa information (see the left hand side part of Figure 6).

Fig. 3. MathLang CGa's colour coding system.

The Core Grammatical aspect (CGa) is a formal language derived from MV and WTT which aims at expliciting the grammatical role played by the elements of a CML text. The structures and common concepts used in CML are captured by CGa with a finite set of grammatical categories. **Terms** represent mathematical concrete objects such as "$\sqrt{2}$" from our example in Figure 2. A **set** is a collection of objects like the set of rationals "$\mathbb{Q}$". A **noun** is a family of mathematical objects that share common characteristics, "number" is an example of a **noun**. **Nouns** could be refined by **adjectives** like "even". A valid document according to CGa's grammar is a succession of **phrases** (**statements**, **declarations** and **definitions**). A **phrase** could be combined in a sequence to form a deduction, this sequencing is called a **block**. One can restrict a **statement** and the **declaration**/**definition** of identifiers to a specific part of the text with a construction called **context** or **local-scoping**. We call **step** an expression which is either a **phrase**, a **block** or a **local-scoping**. MathLang's type system [13] derives typing judgments to check whether the reasoning parts of a document are coherently built. CGa is intentionally elementary and results in a computable low level encoding.

The Text and Symbol aspect (TSa) builds the bridge between a CML text and its grammatical interpretation and adjoins to each CGa expression a string of words and/or symbols which aims to act as its CML representation. The CGa grammar provides computable constructions to represent mathematical reasoning. We added to this strict language information on how each CGa element should be printed on paper or on screen. This makes MathLang's encoding of mathematical texts faithful to traditional mathematical authoring [11]. TSa adds on top of a CML text a new dimension to the document. This dimension is rendered in our examples in Figures 2 and 4 with coloured boxes following the colour coding system of Figure 3.

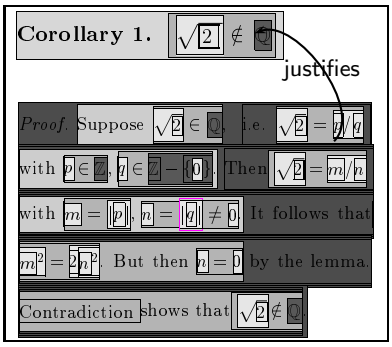


Fig. 4. MathLang's encoding of part of Barendregt's version of Pythagoras' theorem.
This coloured text is the definition and proof of Corollary 1 as written by Barendregt (see Section 1.1). We used TEXmacs and MathLang's plugin for TEXmacs to write this example. The text was automatically checked by our implementation of the MathLang type system [13]. The arrow on top of this text shows the DRa relation between the corollary's definition and its proof.

The coloured boxes shown in Figure 4 are added by the MathLang user himself. We implemented TSa in a *plugin* for the scientific text editor TEXmacs (`http://www.texmacs.org/`). The view of our example shown in Figure 4 was authored using TEXmacs and our *plugin*. The *plugin* transforms the authored document (encoded in TEXmacs' data structure extended with TSa boxes) into MathLang CGa internal XML representation. The document is then checked for CGa grammatical validation. In the rest of this paper we avoid mentioning TSa and concentrate instead on the use one can make of the information held by CGa and DRa.

The Document Rhetorical aspect (DRa) extends the knowledge already computerised in CGa by expliciting the subjective judgments that the author gave on the role some text parts play in the document's structure. At the CGa level, a

document is decomposed into **steps** either put in a sequence or contextualized by the **context** construction. One would encode *division elements* (such as *chapter*, *section*) and *mathematical labeling units* (such as *axiom*, *theorem*, *proof*) by this unique step construction (see the MathLang encoding examples in [12,11,13]). In our example in Figure 4, the proof paragraph is a **step** composed by several sub-**steps**. To enhance flexibility, CGa does not differentiate between these divisions, labels and any other kind of **step**. DRa provides a method to computerise these labels traditionally attributed to chunks of text. These labels and text elements when used in mathematical textbooks or articles give important hints and indications on how to interpret a chunk of text. Moreover, relations between recognised chunks of text sometimes stay implicit in the original document. DRa annotations allow to express such information since the DRa works as an annotation system for the CGa **step**. This annotation system is summarised in Table 1 and consists of:

1. Structural rhetorical role names for *division elements*.
2. Mathematical rhetorical role names for *mathematical labeled units*.
3. A set of *relations* to express relations between *labeled units*/*division elements*.

Description
Instances for the hasStructuralRhetoricalRole *property:* preamble, part, chapter, section, paragraph, *etc.*
Instances for the hasMathematicalRhetoricalRole *property:* lemma, corollary, theorem, conjecture, definition, axiom, claim, proposition, assertion, proof, exercise, example, *etc.*
Relation
Types of relation*:* justifies, subpartOf, uses, exemplifies, inconsistentWith

Table 1. DRa annotations.

Using the DRa annotation system we can capture the role that a chunk of text plays in a document and the relationship that this role imposes on the rest of the document or other chunk of text. This leads to an automatic generation of a dependency graph for the text (see Figure 8) where relations between parts of the text are represented by visible arrows and graph nodes have specified (but not visible) mathematical rhetorical roles. From the annotated narrative feature of a document we receive a *dependency graph* between the chunks of text in a document (e.g. see the left hand side of Figure 6). Those dependencies play an important role in the mathematical knowledge representation. Thanks to those dependencies, the reader can find his own way while reading the original text without the need to understand all its subtleties. Moreover, we will show that these dependencies give the ability to structure the skeleton of a document in a formal language Mizar (see Figure 6).

2.2 Mizar and Formal Proof Sketch

The Mizar system (`http://mizar.org`) is a system for computer checked mathematics [20,21,19,15]. The ongoing development of the Mizar framework, lead by Trybulec since 1973, has resulted in several things: the Mizar system, the Mizar language, the Mizar library and the Mizar software utilities for working with Mizar documents and the Mizar library.

The Mizar language is used for recording mathematics whereas the Mizar system is used for checking the correctness of texts written in this language. The Mizar language is a language suitable for the practical formalisation of mathematics. It is based on first–order logic with free second-order variables. Proofs are written in

the style of natural deduction as proposed by Jaśkowski [9]. The language itself is also an attempt to approximate in a formal way the mathematical vernacular used in publications. On one hand, the Mizar language inherits the expressiveness, naturalness and freedom of reasoning of CML. On the other hand it is formal enough to allow mechanical verification and computer processing.

The Mizar system is accompanied by a library of mathematics – the Mizar Mathematical Library (MML), which is the biggest collection of digitalized mathematical texts verified by computer [24]. MML consists of Mizar documents, which are called *Articles* within the Mizar community. This library is based on two axiomatic *Articles*: `HIDDEN` [4] which consists of built-in notions, and `TARSKI` [22] which presents axioms of the Tarski-Grothendieck set theory. All the other *Articles* of MML are consequences of those axioms and are verified by the Mizar system. The user while writing a new Mizar *Article* reuses the notation, definitions and theorems and other constructs stored in the library. The Mizar system assists the author while formalising new terminology and results. It verifies the claims of the new *Article* and extracts facts and definitions for inclusion into the library. The task of building a rich mathematical library is currently the main effort of the Mizar community. Currently, the library includes 960 *Articles* contributed by 189 authors[1], a number of whom have been active on a long term basis. There is a number of introductory papers and manuals on Mizar [1,20] as well as practical hints for writing Mizar *Articles*.

A Mizar article consists of two parts: the *Environment-Declaration* and the *Text-Proper*. The *Environment-Declaration* begins with `environ` and consists of *Directives*: `vocabularies`, `notations`, `constructors` etc. Roughly, each *Directive* is composed of names of *Articles* from the MML that contain the knowledge required for verifying the correctness of the *Text-Proper*. The *Text-Proper* is a sequence of *Sections*, where each *Section* starts with `begin` and consists of a sequence of theorems and definitions together with their proofs. The division of the *Text-Proper* into *Sections* has no impact on the correctness of the *Article*.

These two parts of the Mizar *Article* are processed by two different programs: *Accommodator* and *Verifier*. The *Accommodator* processes the *Environment-Declaration* and creates the *Environment* in which the knowledge is imported from MML. The *Verifier* has no communication with the library and checks the correctness of the *Text-Proper* using the knowledge stored in the *Environment*.

The Formal Proof Sketch (FPS) notion was introduced by Wiedijk in [25] for declarative systems where the input language of the system is designed to be similar to the language of the informal proofs found in mathematical papers. The FPS notion makes sense for instance for both the Mizar language and the Isar language (used for the Isabelle system). According to Wiedijk:

> A *Formal Proof Sketch* is a text in the syntax of a declarative proof language that was obtained from a full formalization in that language by removing some proof steps and references between steps. The only errors (according to the definition of the proof language) in such a stripped for-

[1] `http://merak.pb.bialystok.pl/` (last accessed on 2007-02-15, MML version: 4.76.959).

> malization should be *justification errors*: the errors that say that a step is not sufficiently justified by the references to previous steps. *[25]*

Even if the above definition states that the FPS version is derived from the full formalization, the process of formalization can start from the informal mathematical document. The process actually consists of two phases: first, one mimics the informal English proof in the formal proof sketch language, second, one fleshes out this formal proof sketch to a full formalization.

The Mizar Formal Proof Sketch (Mizar FPS) is a representation of an informal proof in the formal Mizar language. A text in Mizar FPS is between a fully checkable proof and a statement without any proof at all. It is seen as an incomplete Mizar *Article* that contains holes in the natural deduction reasoning. The application of the Mizar system for a correct Mizar FPS text should result in only one kind of error (the well known `*4` error in the Mizar system), which says that justifications do not necessarily justify the steps. A Mizar Formal Proof Sketch can be completed into a correct fully formalised Mizar *Article* by adding steps and filling essential references for the steps to the proofs. However, it may sometimes happen that the Mizar FPS version needs to be changed to be able to reach full formalisation in the Mizar system. In short, the Mizar FPS version and the full formalisation of an informal text are both written in the same formal language – the Mizar language, and are both checked by the same software – the Mizar system; furthermore, Mizar FPS accepts holes in the reasoning.

3 Narrative features vs. Mizar *Text-Proper* skeletons

The purpose of DRa is to discern explicitly the structure of mathematical knowledge for providing a better encoding of its content, see Figure 8. Using the DRa annotation system we indicate where important mathematical statements start and end. One may argue that this information is visible and we do not need to annotate it explicitly. However, although this information is obvious for a human, it has to be explicitly specified for a computer. As described in Section 2.1, the DRa annotations of a text are used to automatically generate the dependency graph of the text where the relationships between different parts of the text are represented by visible arrows and where the graph nodes have well specified (but not visible) mathematical/structural roles (see the left hand side of Figure 6). We advocate that the DRa annotations of a text and the automatic generation of its dependency graph are useful for the computerisation of a mathematical text because it explicits the narrative features of the text. In this section, we explain how the DRa annotations of a text and its automatically generated dependency graph are used to create a Mizar FPS *Text-Proper* skeleton of the text.

3.1 Transformation hints provided by the dependency graph

Note that the DRa dependency graph (see the left hand side of Figure 6) does not impose any logical correctness. For instance, on our example (see Figure 4), the paragraph labeled proof is related by justifies to a mathematical sentence labeled corollary (see Figure 8). However, this does not imply that the proof indeed proves

the corollary. This latter affirmation is of a different level of importance, and within MathLang it belongs to a different **aspect** than the labeling and relation information. We expect to get an automatic validation of the coherence of relationships drawn in the document (for instance a block of steps labeled proof can not justifies another step labeled axiom).

In this section we give transformational hints which use the dependency graph of a text and the internal representation of the mathematical/structural roles of its nodes, to create a Mizar FPS *Text-Proper* skeleton of the text. It should be noted that we call this stage transformation hints rather than give it a full blown "aspect" status like CGa, TSa or DRa because we have not completed its formalisation/implementation. Currently, we simply give hints to the user. In the future, any implementation of this desired aspect should ask the user, how each relation is used, and in which order the annotated (boxed) text should be.

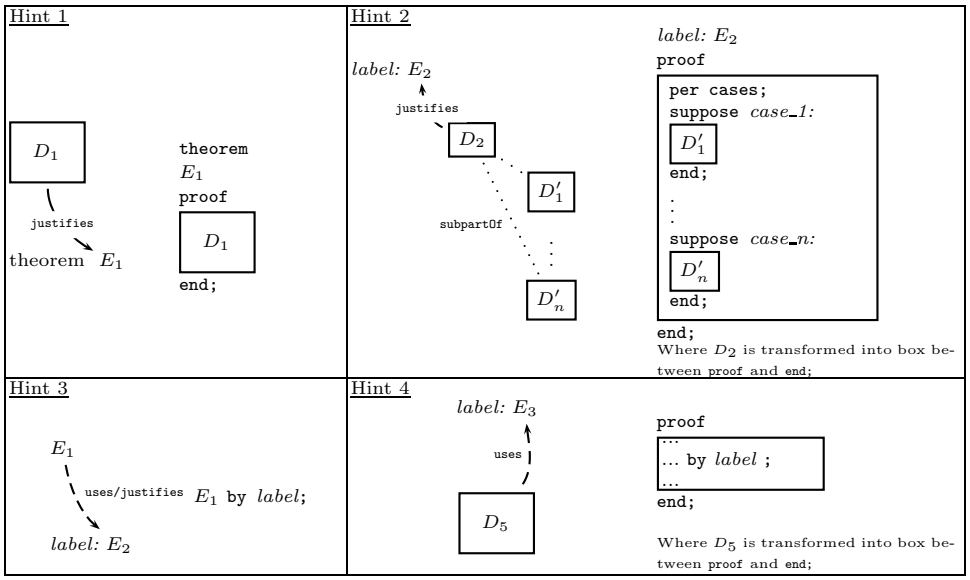


Fig. 5. Four transformation hints provided by the dependency graph.

Figure 5 lists four hints that we use to transform our main example. In each of these hints, a dependency graph (on the left hand side of the hints) is transformed into a Mizar specific structure (on the right hand side). For example, in hint 1, if we annotate a box, let say E_1, as a theorem, it could be transformed into Mizar syntax as: `theorem` E_1. Moreover, if we say that a box $\boxed{D_1}$ has the mathematical role proof, then we can transform it into: `proof` $\boxed{D_1}$ `end;`. Moreover, since a block of steps having the mathematical role proof is in relation justifies with a single statement, we can say that this is a particular *Proof Justification* in Mizar, which is transformed into a specific form like the right hand side of hint 1.

Hint 2 deals with proof by cases. If $\boxed{D_2}$ is in relation justifies with a statement E_2, and consists of the parts $\boxed{D'_1}$,..., $\boxed{D'_n}$, then we can give a user hint that this is a *Proof Justification* in which reasoning is done by all the cases.

In hint 3, the relation uses, could express a Mizar *Straightforward-Justification*, for instance, if a sentence E uses or justifies sentence E, then we can inform the user that this corresponds to a Mizar *Straightforward-Justification*.

In the dependency graph of hint 4, block $\boxed{D_5}$ uses statement E_3. Here, we transform block $\boxed{D_5}$ into a specific Mizar *Proof* block, which contains an expression with *Straightforward-Justification* to statement E_3.

3.2 Applying the transformation hints to our example

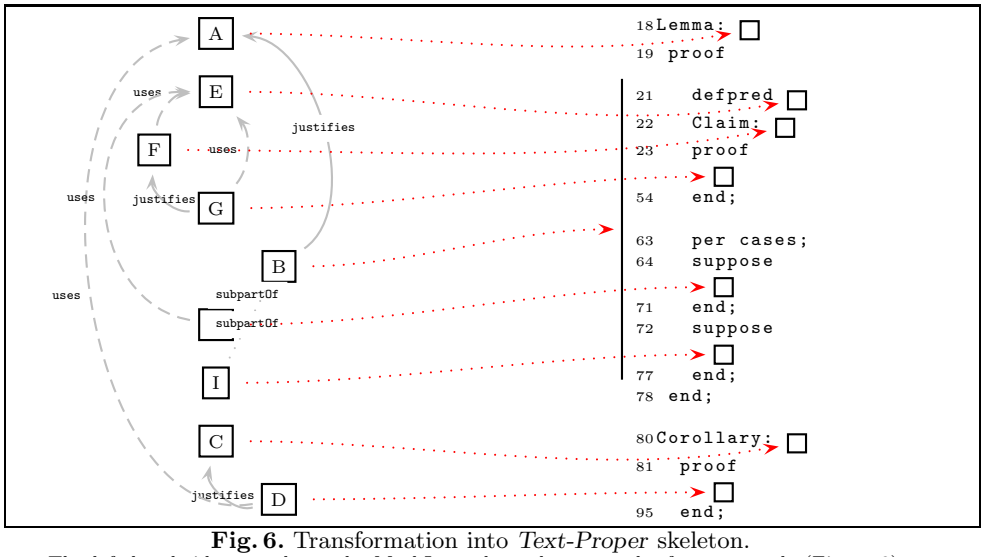


Fig. 6. Transformation into *Text-Proper* skeleton.
The left hand side reproduces the MathLang dependency graph of our example (Figure 8). On the right hand side we show the Mizar *Text-Proper* skeleton of the same example. The arrows from left to right shows how the MathLang dependency graph gives hints on how to build the Mizar *Text-Proper* skeleton.

Using the transformation hints of Figure 5, we can transform the dependency gaph produced for our main example (see the left hand side of Figure 6) into a proper structure of the *Text-Proper* part of our Mizar FPS (see the right hand side of Figure 6). As already discussed, this transformation is not done automatically. It is our intention in the near future, to formalise and implement the transformation into a further aspect of MathLang.

4 Building parts of Mizar FPS from a grammatically annotated document

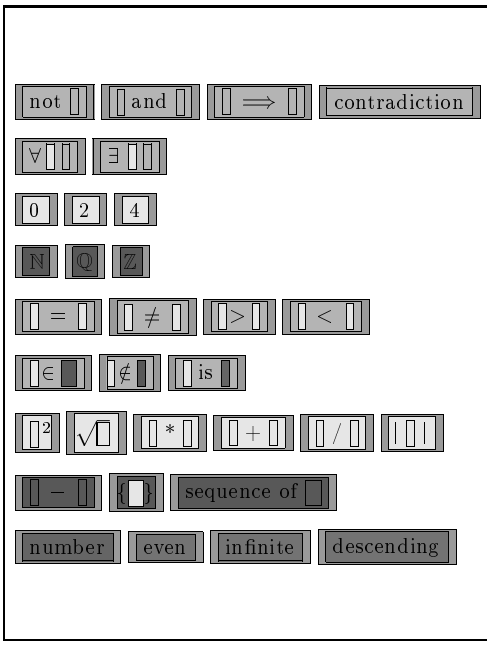
4.1 The document's background knowledge

When a document has been properly encoded in CGa, all the notions used in the document would be properly declared with the appropriate CGa grammatical categories. This results in a list which declares the identifiers used in the document and which form an important part of the background knowledge required to understand it. For example, the arithmetic operations plus, times or square root are not defined in our main example (see Figure 8) but are assumed to be known by the reader. At the CGa encoding level of our example, these arithmetic operations

need to be declared at the start of the encoded document. The common way to do so is to start the document with a **context** containing the list of declarations of all these symbols and notions (see Figure 7). At the DRa level, we identify this list of declarations as a **preamble** by annotating the CGa paragraph containing the left hand side of Figure 7 by:

[description hasStructuralRhetoricalRole="preamble"]

In our example, the identifier even (line 28 of Figure 7) is in the **preamble** because it is used in the original document but not defined. We expect these identifiers to have a proper definition outside the original text. One can understand them with a good mathematical background or with access to the background literature (we omit here the way MathLang adopts to refer to external documents). Each of these externally defined identifiers has to be declared.




```
4 { not(stat): stat;
5   and(stat,stat): stat;
6   implies(stat,stat): stat;
7   contradiction(): stat;
8   forall(dec('x), stat) : stat;
9   exists(dec('x), stat) : stat;
10  0: term; 2: term; 4: term;
11  N: set; Q: set; Z: set;
12  =(term,term): stat;
13  neq(term,term): stat;
14  >(term,term): stat;
15  <(term,term): stat;
16  in(term,set): stat;
17  notin(term,set): stat;
18  is(term,noun): stat;
19  sq(term): term;
20  sqrt(term): term;
21  *(term,term): term;
22  +(term,term): term;
23  /(term,term): term;
24  abs(term): term;
25  subtraction(set,set): set;
26  one_element_set(term): set;
27  number: noun;
28  even: adj;
29  sequence(set):noun;
30  infinite:adj;
31  descending:adj; };
```

Fig. 7. Preamble of MathLang's encoding of Pythagoras' theorem
The left hand side presents the **preamble** as shown by TSa whereas the right hand side shows the corresponding lines in the automatically generated CGa (printed using CGa's abstract syntax as defined in [13]).

In Mizar, the *Environment* plays a similar role to the MathLang **preamble** by describing the background knowledge of the *Article*, however, there is one subtle difference. Namely, the *Environment* in Mizar lists the MML entries that have to be loaded prior to any analysis of the *Text-Proper* part of the *Article* (see Listing 1.1 for our example's *Environment*). The *Articles* to be loaded contain, among other things, the notations and definitions that are used in the *Text-Proper* part. This gives a slightly different constraint to the authoring: in MathLang the author simply needs to list the external identifiers, whereas in Mizar the author needs to select the background MML literature to use. The MathLang CGa is more concerned

with the "visible" external identifiers (CGa is about the grammatical completeness and therefore only needs the grammatical signature of each identifier) but Mizar needs to have a complete semantic and logic background (with *Definitions* or *Proofs* associated to each identifier).

Listing 1.1. Mizar FPS *Environment*

```
environ
 vocabularies INT_1, SQUARE_1, MATRIX_2, IRRAT_1, RAT_1, ARYTM_3, ABSVALUE,
   SEQM_3, FINSET_1;
 notations INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1, XCMPLX_0,
   INT_2, SEQM_3, FINSET_1, REAL_1, PEPIN;
 constructors INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1,XCMPLX_0,
    INT_2, SEQM_3, FINSET_1, PEPIN;
 requirements SUBSET, NUMERALS, ARITHM, BOOLE, REAL;
 registrations XREAL_0, REAL_1, NAT_1, INT_1;
```

The **preamble** of the CGa encoding is crucial in the migration process of encoding into a Mizar FPS version of the text. We treat the information of the **preamble** as a subset of the Mizar *Environment*. In Mizar FPS we use the same symbols and identifiers that were explicitly introduced in the CGa, although some of them have different spelling. We have to remember that at some point we can acquire the CGa encoding of a mathematical text, which could contain identifiers or symbols that have not been defined in the MML yet. In such case we have to define those identifiers in the *Text-Proper* part of the Mizar FPS and introduce their names in the associated *Vocabulary* file. This situation requires much more investigation in the future.

In this section, we use the **preamble** to build two parts of the Mizar FPS *Environment-Declaration*, namely *Directives*: `vocabularies` (which consist of MML entries that store symbols used in the *Text-Proper* part of the *Article*) and `notations` (which consist of MML entries that store notions of symbols used in the *Text-Proper*). The information in the **preamble** gives hints how the identifiers could correspond to Mizar counterparts. By filling only those two *Directives* in the *Environment-Declaration*, we can check the *Text-Proper* part of the Mizar FPS in terms of "grammatical correctness". After those *Directives* are fully filled, we call the Mizar system with a special option (i.e. `accom -p $PATH/file_name.miz` and `verifier -p $PATH/file_name.miz`). If the Mizar system does not return any error, then the *Text-Proper* part of the *Article* is "grammatically correct" according to the Mizar grammar, and the symbols and their *Formats* that are used in the *Text-Proper*. The *Format* describes the number of arguments and the order (infix, prefix or postfix) in which the arguments of a *Constructor Symbol* may be used. Although, this partially filled *Environment-Declaration* allows to check the "grammatical correctness" of the *Text-Proper*, the *Environment-Declaration* needs to be more fully filled to achieve a proper Mizar FPS where the only errors are *Justification* errors.

We do not show in the paper how to build the proper *Environment* and how to search the MML. We only express briefly that identifiers in the CGa correspond more or less to Mizar *Items*. Such correspondence gives the overall idea and hints as to what kind of *Items* we have to search for in MML or to introduce in the *Text-Proper* in case they are not yet defined in MML.

4.2 Mathematical identifiers and their formal counterparts

Mathematical conjunctions. In the **preamble** created in our example (see Figure 7) one can find the introduced identifiers that play the role of statement conjunctions, e.g. *implies* or *and*. These are usually reserved words and terminal in the Mizar Language, and they are used to form *Formulas*, see the table below.

CML	CGa	Mizar
or	*id* (stat, stat): stat;	*Formula* `or` *Formula*
and		*Formula* `and` *Formula*
implies	where the identifier's name *id*	*Formula* `implies` *Formula*
iff	is choosen by the user, e.g. lines 4–7	*Formula* `iff` *Formula*
not	in Figure 7.	`not` *Formula*

Declarations of these conjunctions are presented in CGa as identifiers that take two arguments of type `stat` and return the same type `stat`. This typing information allows us to assume that these identifiers are expressed as Mizar reserved words, which have the same spelling as in CML.

Binders like '∀', '∃' or '$\sum$' are indispensable parts of CML. In CGa, the user has to declare them (see figure 7). The CGa is flexible and allows any kind of binder. The Mizar user, if he wants to introduce a new binder, has to do so in an indirect way, although the syntax for Mizar binders was proposed in [23]. Nonetheless, the Mizar language offers the two most essential binders: ∀ and ∃ which are given as *Quantified-Formula* in the Mizar syntax (see the table on the right).

CML	∀, ∃
Possible CGa	8 `forall(dec('x), stat) : stat;` 9 `exists(dec('x), stat) : stat;`
Mizar	*Quantified-Formula* = `for` *Qualified-Variables* [`st` *Formula*] (`holds` *Formula* | *Quantified-Formula*) | `ex` *Qualified-Variables* `st` *Formula* (`holds` *Formula* | *Quantified-Formula*)

Functions identifiers in CGa are closely related to *Functor-Definitions* in Mizar. The information that we gain from the CGa encoding is the number of arguments and their weak input and output types. For instance `sq` is a function which takes one argument and return a value with the same type as argument (see the table on the right). However, it is common knowledge that this function corresponds to the mathematical function *square* (usually written as ^{2} in CML). With this information, we can search MML to find the appropriate Mizar function definition counterpart, which is introduced as *Functor-Definition*. Similarly to such specific CGa identifiers, functions in Mizar have to define the types of their arguments and the type of their results. Mizar *functors* are constructors of (atomic) term, i.e. applied to a (possibly empty) list of terms they create a term.

CML	_ ^{2}
Possible CGa	19 `sq(term): term;`

Mizar	`definition let x be complex number;`
	`  func x^2 equals`
	`:: SQUARE_1:def 3`
	`  ...`
	`end;`

CML	_ \ _
Possible CGa	25 `subtraction(set,set): set;`

Mizar	`definition let X,Y be set;`
	`  func X \ Y -> set means`
	`:: XBOOLE_0:def 4`
	`  ...`
	`end;`

Predicates. In the CGa encoding some identifiers play the role of predicates which take arguments of type term or set and return stat. In Mizar these identifiers are given in terms of *Predicate-Definition* (see the tables on the right). *Predicates* in Mizar are constructors of atomic formulas, can have several predefined properties (e.g. `symmetry`, `reflexivity` etc.) and define the type of their arguments. We claim that some CGa identifiers (with input types: term or set, and output type stat) correspond more or less to Mizar *Predicates*.

CML	_ ∈ _
Possible CGa	16 in(term,set): stat;

Mizar	notation let a,b be ext-real number;
	antonym b < a for a <= b;
	end;

CML	_ < _
Possible CGa	15 <(term,term): stat;

Mizar	notation let a,b be ext-real number;
	antonym b < a for a <= b;
	end;
	where the original predicate is defined as follows:
	definition let x,y be ext-real number;
	pred x <= y means
	:: XXREAL_0:def 5
	....
	end;

Nouns in CML are abstract concepts that classify objects according to their characteristics. The CGa notion of **noun** corresponds to the notion of *Types* in Mizar. *Types* in Mizar are defined using either *Mode-Definitions* or *Structure-Definitions*. For example, the identifier number declared as a **noun** in CGa corresponds to *Mode* in Mizar (see the table on the right). One can also define a **noun** in CGa by giving its features with a **step**. This corresponds to the *Definiens* inside either *Mode-Definition* or *Structure-Definition* which helps to find within MML a proper *Type*. For example, a noun description of the identifier group in CGa (see the example in [13]) could help to identify the *Type* `Group` in Mizar.

CML	*number*
Possible CGa	27 number: noun;
Mizar	notation synonym number for set; end; where set is the primitive type (i.e. the widest type) in Mizar introduced as a *Mode-Definition* in the article HIDDEN

Adjectives are another essential part of CML. The CGa notion of adjectives corresponds to the notion of *Attributes* in Mizar. Mizar *Attributes* are defined using *Attributes-Definitions*. For example, the identifier even declared as an **adjective** in CGa corresponds to the *Attribute* in Mizar (see the table on the right). Furthermore, adjectives in CML and CGa are used to modify the characteristics of a noun. Similarly, in Mizar we use *Adjectives* to refine *Types* [2].

CML	*even*
Possible CGa	28 even: adj;
Mizar	definition let i be number; attr i is even means :: ABIAN:def 1 ... end;

Other identifiers. In CGa we have declared some identifiers to be terms, e.g. 0, 2, 4 (see line 10 of Figure 7), whereas in Mizar they are treated as *Numerals*, which have not been introduced inside MML.

Other identifiers, that have been introduced while computerising our example in CGa, are sets, i.e. N, Q, Z (see line 11 of Figure 7). These represent well known

mathematical sets of numbers, i.e. $\mathbb{N}$, $\mathbb{Q}$, $\mathbb{Z}$ respectively. In the Mizar Mathematical Library these sets are introduced as *Functors* (via *Functor-Definitions*) with empty lists of terms, using the symbols: `NAT`, `RAT`, `INT` respectively.

4.3 Transforming the document building steps

As already mentioned (see Section 2.1), in MathLang we present **phrase**, **block** and **local-scoping** in terms of **step** and treat a block as a single **step** composed of a sequence of **statements**. Moreover, the CGa **preamble** gives hints how the identifiers should be translated in Mizar and in which Mizar *Format* (i.e. which Mizar symbols and the place and number of arguments). This information is used to put Mizar symbols inside *Formulas*.

In this section we show using a number of examples, how particular **steps** of our main example encoded in CGa are represented in the Mizar language. Although, we do not give hints how each CGa **step** could be transformed into the Mizar language, we show some ideas through small examples.

Atomic statements in CGa correspond to Mizar's *Formulas*.

CML	... that m^2 is even, but ...
Possible CGa	`44 is(sq(m), even number);`
Mizar	`28 m^2 is even ;`

Blocks in MathLang and in Mizar express a sequence of statements/steps: $\{step_1, \ldots, step_n\}$ (see the example below). In MathLang, if a block is accompanied with a particular mathematical rhetorical role (using the DRa annotation system), it could be transformed into a Mizar specific structure. For instance, if a MathLang **block** is annotated as proof using the DRa, it will still be treated as a sequence of **steps** within the CGa. However, in Mizar, it is transformed to a special *Proof Justification* : `proof` *Reasoning* `end;` (see Section 3).

CML	Possible CGa	Mizar

... $m^2 = 2n^2$. But then $n = 0$ by the lemma. Contradiction shows that $\sqrt{2} \in \mathbb{Q}$.	`89 {`
	`90 =(sq(m),*(2,sq(n)));`
	`91 Lemma |> =(n,0);`
	`92 contradiction;`
	`93 };`
		`89 m^2 = 2*n^2;`
		`90::> *4`
		`91 n = 0 by Lemma;`
		`92::> *4`
		`93 hence contradiction;`
		`94::> *4`

Contexts. In CGa we use **local-scoping**, i.e. $step_1 \rhd step_2$ which makes the declarations, definitions, and assertions inside $step_1$ available inside $step_2$. This allows to build any kind of **context** for another statement or part of the document. For example, we can use **local-scoping** to introduce a new local predicate. In Mizar, this is introduced as a private predicate (via *Private-Predicate-Definition*), and does not need any kind of context (see the table below).

CML	Define on $\mathbb{N}$ the predicate: $P(m) \iff \exists n.m^2 = 2n^2 \ \& \ m > 0$.
Possible CGa	`38 { m1: N; } |>` `39 P(m1):= exists(n1: N, and(=(sq(m1),*(2,sq(n1))),>(m1,0)) );`
Mizar	`21 defpred P[Nat] means ex n being Nat st $1^2 = 2*n^2 & $1 > 0;`

Another possible way of presenting the **local-scoping** usage is to make assumptions into a context to be used in the reasoning block (see the table below). Based on such specified assumptions we can provide further deduction.

We can use *local-scoping* to introduce a new (local or global) variable with a statement expressing some property of this variable. In Mizar this is called *Choice-Statement* (see the table below). In such **context** we introduce a local variable, which is actually bounded with the skeleton of the proof, in the sense that it doesn't change the proof. We have treated this as referring to an available proposition (probably being a consequence of reasoning, for instance definition unfolding) "`ex x being T st P[x]`", where `x` is *Term*, `T` is *Type* and `P` is *Predicate*. From this we can write "`consider a being T such that P[a]`". We do this when we want to reuse the introduced variable in further reasoning steps.

CML	suppose $m^2 = 2n^2$
Possible CGa	68 { =(sq(m),*(2,sq(n))); } |> 69 { ... 76 };
Mizar	62 `assume A0: m^2 = 2*n^2;`

CML	So $m = 2k$ and we have
Possible CGa	46 { k: Nat; =(m,*(2,k)); } |> 47 { ... 50 };
Mizar	32 `consider k being Nat such that m = 2*k;`

The above listed examples of the **local-scoping** construct show only ideas how it could be used when computerizing mathematics. The usage of this construction is not limited to these examples and gives a lot of freedom and flexibility when representing mathematical expressions in MathLang. Therefore it is difficult to propose one transformation hint for the Mizar corresponding structure.

5 Different formalisation paths

The direct path from CML to Mizar – (© of Figure 1). When transforming a CML text directly to Mizar, a number of facts need to hold:

1. The user needs to be a specialist in the Mizar system (including MML and its search engines: MML Query or the `grep` tool). Furthermore, the user's expertise needs to encompass both mathematics and computer science. In fact, even if a Mizar *Article* resembles a CML text, Mizar is much more closer to declarative programming languages (e.g. Pascal).
2. CML texts can be ambiguous, and the user needs to find and clarify those ambiguities when formalising the text.
3. The Mizar user needs to interpret the text, to find the meaning of each part of the document, and to present it in a formal way.
4. Although the Mizar user has a choice (he may first present parts of the text in the Mizar language, or start from a single statement and look in MML for knowledge that allows the presentation of this statement in the Mizar language, and then to rewrite it; if the statement is accompanied with the proof, maybe the user could fully formalize the proof, and move forward to rewriting the rest of the document within Mizar), there is a common way to translate a text into Mizar which is as follows:

- Start from a single statement, write it in Mizar.
- If the statement is accompanied with the proof then fully formalize the proof, if the statement is a definition then define it in Mizar with the proper definiens and prove Mizar specific conditions for the definition.
- Then, move forward within the CML and Mizar translation and finally, reveal the rest of the reasoning structure of the CML text in Mizar.

The path from CML to Mizar FPS to Mizar – (ⓑ-ⓔ of Figure 1). When transforming a CML text to Mizar FPS and then to Mizar, a number of facts needs to hold:

1. The user needs to be a Mizar specialist as well, and requires similar amount of knowledge as the user who follows the direct path ⓒ to Mizar.
2. The difference (see point 4 from the above list) is that first, the user needs to structure the whole CML text in Mizar FPS. At this stage the user actually does not stop to fully formalize a particular definition or theorem. Although such a choice is possible, Mizar FPS is not meant to do that.
3. After structuring the CML text in Mizar FPS, the user needs to complete the formalisation by filling all the gaps in the reasoning (i.e., filling the holes in sentences that were labelled with the error `*4` by the Mizar system.)
4. Since the level of ambiguity of a text is the same as in the direct path, the user needs to carry out the same amount of work as above.

At this stage we could say that although step ⓒ might lead to the same result of steps ⓑ-ⓔ, the work done via ⓑ-ⓔ can be more enjoyable for the Mizar user and a bit easier.

Our proposed path from CML to MathLang to Mizar FPS to Mizar – (ⓐ-ⓓ-ⓔ of Figure 1). When transforming a CML text to MathLang, then to Mizar FPS and then to Mizar, a number of facts needs to hold:

1. The first part of the path (ⓐ) is done by a mathematician, who does not require a lot of MathLang knowledge when annotating the text with CGa grammatical categories or assigning the relationships between different parts of the text and the mathematical or structural rhetorical roles different mathematical entities play. The mathematician simply reveals his understanding of the text. This annotation gives some advantages:
   - It explicits all the identifiers used in the text together with a number of arguments and their weak input and output types.
   - It resolves some ambiguities of the text.
   - It allows the document to be grammatically validated via the automatic CGa checker.
   - It specifies the roles of the important chunks of the text, and expresses dependencies between them.
   - The dependencies of the text parts and the internal information about the roles entities play, allow the automatic generation of a dependency graph which gives the reasoning structure of the text.
   - At this point the work of the mathematician is finished.

2. The Mizar specialist takes the tagged document within MathLang and transforms it into Mizar FPS (part ⓓ of the path). Here, the user needs to be a Mizar specialist and to have the same Mizar knowledge as in the direct path and the one via Mizar FPS.
3. However, at step ⓓ, the user has a structure of the CML text (tagged by the mathematician's understanding of the text and the DRa and CGa **steps**) which helps him to build the skeleton of Mizar FPS. Secondly, all the used identifiers, with the number of their arguments, are stored in one place (the DRa explicit annotation of the **preamble**), and could be reused to find counterparts in Mizar MML and to build parts of the *Environment*. The user also gains from resolved ambiguities of the CML text within MathLang. We believe that this makes the work for the Mizar user a lot easier.
4. At this stage we could say that although step ⓑ might lead to the same result of steps ⓐ-ⓓ, the work done via ⓐ-ⓓ gives an active role to the mathematician in the computerisation and allows the mathematician's computerisation to give a number of useful hints to the Mizar user to create the Mizar FPS skeleton and the Mizar FPS version of the text.

We believe that it is worth following our proposed path. Not every mathematician is interested in fully formalising mathematical texts. Sometimes one may just want a partial formalisation, or even to formalise and verify the correctness of one particular theorem/proof. We believe it is too taxing on mathematicians to ask them to learn the language and specific logic of a proof checker. The advantage of using MathLang as an intermediate step in the proposed path towards Mizar FPS is a guidance for non expert-authors. This guidance mainly helps to extract from the original text an indication of the required background knowledge and an abstraction of the reasoning structure of the text.

6 Conclusions, Related and Future Work

Conclusions. We have presented in this paper our MathLang approach to encoding mathematics on computers. This approach defends the idea that computerisation should come before any formalisation. We briefly showed how MathLang could be used as a useful computerisation tool for the ordinary mathematician who can use it to edit his text (as if he was using LaTeX) and then get a number of automated features and programs that enable him to to create a number of computerised versions of the text. These computerised versions have useful information about the original text, and are then used by the Mizar expert to create first a Mizar FPS version and then a fully formalised Mizar version.

- The main advantage of using MathLang as a mathematical framework is a clear guidance for the non-expert author. This guidance mainly helps to extract from the original text different aspects of mathematical knowledge at different phases of its computerisation.
- The DRa annotation gives useful hints how the skeleton of the Mizar FPS *Article* can be built.

– The CGa **preamble** is treated as a subset of the Mizar *Environment*. CGa identifiers have corresponding counterparts within the Mizar library or could be introduced as Mizar specific *Definitional-Items*. This gives hints about the way we could transform CGa identifiers into their counterparts in Mizar and for which kind of *Symbols* and *Formats* we need to search in MML.

At the time of writing the paper, we used the most recent Mizar system: Mizar system version: 7.8.03 and the MML version: 4.76.959. Due to page constrains we do not attach the complete formalisation of our example in Mizar. However, it is available on-line: `http://www.macs.hw.ac.uk/~retel/pythagoras/`.

Related work. Geleijnse [7] compared WTT and Mizar, presented CML examples in both WTT and Mizar and gave a correspondence between WTT and Mizar identifiers. His main approach was based on comparing these two languages. Our approach is completely different (although of course we are indebted to all the progress in Automath, MV, WTT, FPS and Mizar). Even though inspired by MV and WTT, MathLang's CGa has moved towards an automatically generated structure obtained from the mathematician's editing of the text at the TSa level where the mathematician types his text easier than using LaTeX (in fact, we can claim that this stage is as easy as if the mathematician is writing his text on paper). TSa also gives the mathematician editing features that allows him to assign mathematical, structural and rethorical roles, to entities and chunks of the text and relationships between these chunks. The automatic programs of MathLang create not only the CGa version of the text but also the dependency graph of the text which is then used to create a Mizar FPS *Text-Proper* version of the text. Our path ⓐ-ⓓ-ⓔ of Figure 1 is fully worked out and offers the user much computerised help along the way, and a number of well-formulated hints used in the gradual computerisation and formalisation of the text from the original CML version to a number of computerised versions (CGa, DRa, TSa) followed by a Mizar skeleton followed by Mizar FPS and full Mizar versions. Furthermore, although the de Bruijn path principle (see Section 2.1) has played an influential role in this research, the various levels (or aspects, or stages) of our proposed path are new. Another approach which follows the de Bruijn path principle is discussed by Jojgov and Nederpelt in [10]. However their description of a path from CML to type theory via WTT and type theory with open terms (TTOT) starts from a WTT-text which differs from (but represents) the original CML-text, and then takes the WTT-text into a TTOT version and later into type theory. Our approach starts from the original CML-text (which is the input given by the mathematician into MathLang TSa). The process of moving from the CML-text input into a Mizar FPS skeleton is supported by a number of automated MathLang programs and transformed into the Mizar FPS full version by the Mizar specialist and fully checked by the Mizar system.

Future Work. MathLang is an ongoing project. As we have seen, the theoretical formalisation and computer implementation of the first three aspects provided a number of useful tools that automatically generate a number of computerised versions of the text each used for a different purpose and each enjoys a different level of formality. As we have also seen, further aspects need to be formalised. For example, it is important to have an aspect that transforms the dependency graph

into a Mizar *Text-Proper* skeleton (currently we only provide hints for doing so). It is also important to study in depth the stage where a Mizar FPS version is fully formalised in Mizar. A number of issues need to be investigated:

- How to employ search engines (like `grep` or semantic mining MML Query) to look up MML in order to find a proper Mizar counterpart for an identifier used in a CML text and explicitly stated in its MathLang CGa version.
- How the MathLang **noun** description construction could be reused to find a counterpart in MML or to define either a *Mode* or a *Structure*.
- How to deal with the freedom that MathLang gives while computerising a common mathematical document.
- We also need to express the hints for transforming a dependency graph into a Mizar FPS *Text-Proper* skeleton, in terms of formal rules which we aim to prove correct and to implement. Moreover, our aim is to start building a computer tool which will support the Mizar specialist with the migration process from a CML+MathLang document to Mizar FPS.

References

1. *Mizar Manuals*. `http://mizar.org/project/bibliography.html`.
2. G. Bancerek. On the structure of Mizar types. *ENTCS*, 85(7):1–17, 2003.
3. Henk Barendregt. *Informal*, page 10. Volume 3600 of Wiedijk [26], 2006.
4. Library Committee. Mizar built-in notions. *Journal of Formalized Mathematics*, Axiomatics, 1989. `http://mizar.org/JFM/Axiomatics/hidden.html`.
5. N. G. de Bruijn. Checking mathematics with computer assistance. *Notices of the American Mathematical Society*, 38(1):8–15, 1991. Available at the Automath Archive: `http://automath.webhop.net/` Brouwer Institute in Nijmegen and the Formal Methods section of Eindhoven University of Technology.
6. N.G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In *Workshop on Programming Logic*, 1987.
7. G. Geleijnse. Comparing two user-friendly formal languages for mathematics: Weak type theory and Mizar. Master's thesis, Technische Universiteit Eindhoven, May 2004.
8. Godfrey Harold Hardy and Edward Maitland Wright. *An introduction to the Theory of Numbers*. Oxford University Press, 5th edition, April 1980.
9. S. Jaśkowski. On the rules of supposition in formal logic. *Studia Logica*, 1934.
10. G. Jojgov and Rob Nederpelt. A path to faithful formalizations of mathematics. In MKM '04 [16], pages 145–159.
11. Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Flexible encoding of mathematics on the computer. In MKM '04 [16], pages 160–174.
12. 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.
13. Fairouz Kamareddine, Manuel Maarek, and J. B. Wells. Toward an object-oriented structure for mathematical text. In Michael Kohlhase, editor, *Mathematical Knowledge Management, 4th Int'l Conf., Proceedings*, volume 3863 of *Lecture Notes in Artificial Intelligence*, pages 217–233. Springer, 2006.
14. Fairouz Kamareddine and Rob Nederpelt. A refinement of de Bruijn's formal language of mathematics. *J. Logic Lang. Inform.*, 13(3):287–340, 2004.

15. R. Matuszewski and P. Rudnicki. Mizar: the first 30 years. Volume 4 of *Mechanized Mathematics and its Applications*, pages 3–24, March 2005. `http://mizar.org/people/romat/MatRud2005.pdf`.
16. *Mathematical Knowledge Management, 3rd Int'l Conf., Proceedings*, volume 3119 of *Lecture Notes in Computer Science*. Springer, 2004.
17. R. P. Nederpelt and F. Kamareddine. Formalising the natural language of mathematics: A mathematical vernacular. In *4th Int'l Tbilisi Symp. Language, Logic & Computation*, 2001.
18. 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.
19. P. Rudnicki. An overview of the Mizar project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, 1992.
20. P. Rudnicki and A. Trybulec. On equivalents of well-foundedness. An experiment in Mizar. *Journal of Automated Reasoning*, 23(3–4):197–234, 1999.
21. A. Trybulec. The Mizar logic information language. *Studies in Logic, Grammar and Rhetoric*, 1, 1980. Bialystok.
22. Andrzej Trybulec. Tarski Grothendieck set theory. *Journal of Formalized Mathematics*, Axiomatics, 1989. `http://mizar.org/JFM/Axiomatics/tarski.html`.
23. F. Wiedijk. A proposed syntax for binders in Mizar. `http://www.cs.ru.nl/~freek/notes/`.
24. F. Wiedijk. Comparing mathematical provers. In *Mathematical Knowledge Management, 2nd Int'l Conf., Proceedings*, volume 2594 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2003.
25. F. Wiedijk. Formal proof sketches. In *Proceedings of TYPES'03*, volume 3085 of *LNCS*, pages 378–393. Springer-Verlag, December 2004.
26. F. Wiedijk, editor. *The Seventeen Provers of the World, foreword by Dana S. Scott*, volume 3600 of *LNCS*. Springer Berlin, Heidelberg, 2006.

A Original and DRa-annotated text of our example

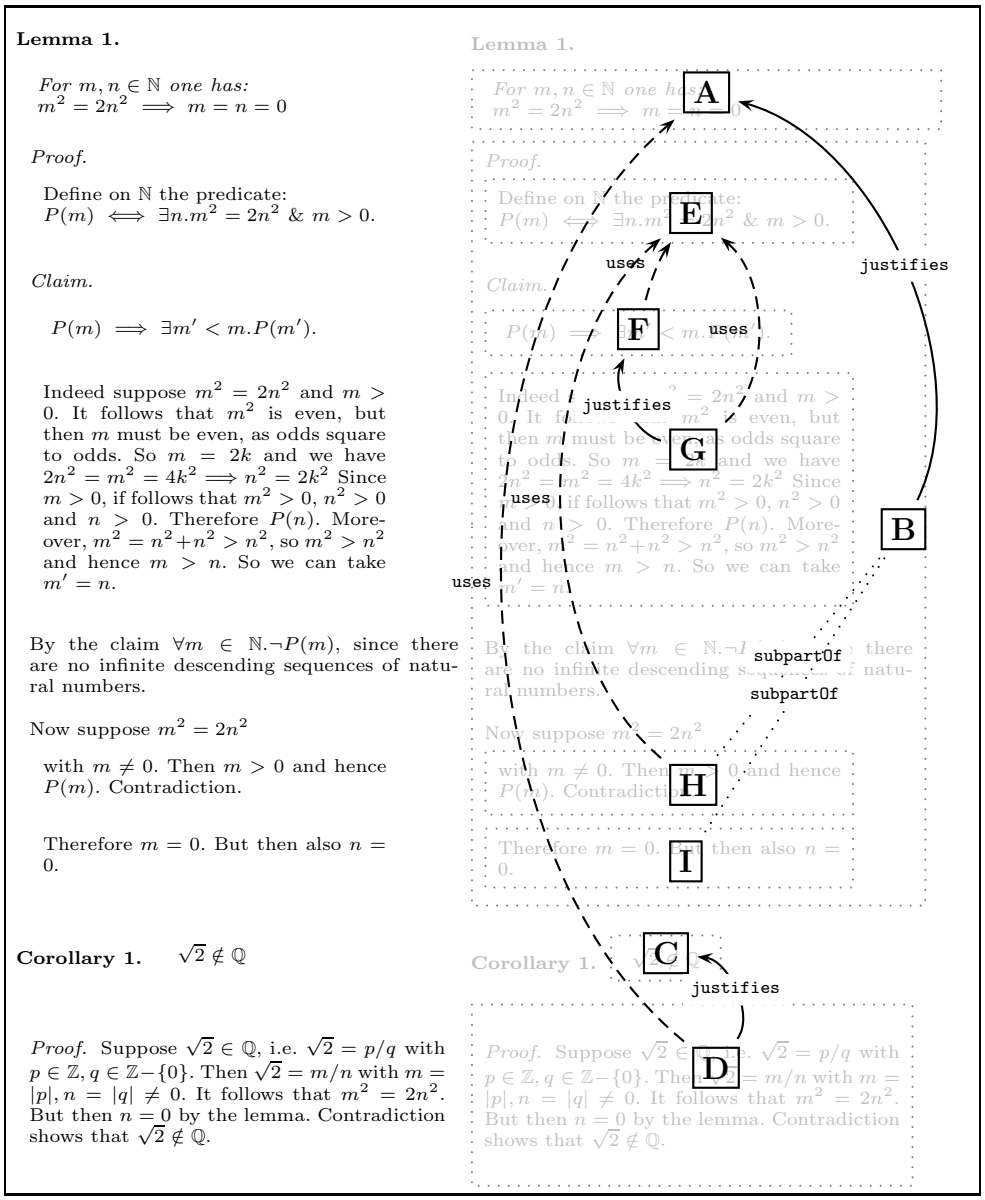


Fig. 8. Barendregt's version (without and with dependency graph) of the proof of the Pythagoras' theorem

The original text of Barendregt's version[3] of the proof of the Pythagoras' theorem is reproduced on the left hand side. The right hand side of the figure shows the automatically generated dependency graph for the text where relations between parts of the text are represented by visible arrows and graph nodes have specified (but not visible) mathematical structural roles.

B The Mizar Formal Proof Sketch presentation

Listing 1.2. Encoding of the example from Figure 8 in the Mizar FPS

```
:: This file is verified with the system version:
:: Mizar verifier= 7.8.03,MML = 4.76.959
::
:: Created by Krzysztof Retel {retel@macs.hw.ac.uk}

environ
 vocabularies INT_1, SQUARE_1, MATRIX_2, IRRAT_1, RAT_1, ARYTM_3, ABSVALUE,
   SEQM_3, FINSET_1;
 notations INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1, XCMPLX_0,
   INT_2, SEQM_3, FINSET_1, REAL_1, PEPIN;
 constructors INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1,XCMPLX_0,
    INT_2, SEQM_3, FINSET_1, PEPIN;
 requirements SUBSET, NUMERALS, ARITHM, BOOLE, REAL;
 registrations XREAL_0, REAL_1, NAT_1, INT_1;
begin


Lemma:  for m,n being Nat holds m^2 = 2*n^2 implies m = 0 & n = 0
 proof
   let m,n being Nat;
   defpred P[Nat] means ex n being Nat st $1^2 = 2*n^2 & $1 > 0;
   Claim: for m being Nat holds P[m] implies ex m' being Nat st m' < m & P[m']
   proof
     let m being Nat;
     assume P[m];
     then consider n being Nat such that
     m^2 = 2*n^2 & m > 0;
     m^2 is even ;
::>            *4
     m is even;
::>          *4
     consider k being Nat such that m = 2*k;
::>                                      *4
     2*n^2 = m^2
::>            *4
          .= 4*k^2;
::>             *4
     then n^2 = 2*k^2;
     m > 0 implies m^2 > 0 & n^2 > 0 & n > 0;
::>                                       *4,4,4
     then P[n];
::>          *4,4
     m^2 = n^2 + n^2;
::>                *4
     n^2 + n^2 >  n^2;
::>                 *4
     then m^2 > n^2;
::>               *4
     then m > n;
::>           *4
     take m' = n;
     thus thesis;
::>            *4,4
   end;
   A2: for k being Nat holds not P[k]
   proof
     not ex q being Seq_of_Nat st q is infinite decreasing by Claim;
::>                                                              *4
     hence thesis;
::>             *4
   end;
   assume A0: m^2 = 2*n^2;
   per cases by A0;
```

```
64   suppose B1: m <> 0;
65     then m > 0;
66::>           *4
67     then P[m] by B1;
68::>             *4
69     then contradiction by A2;
70     hence thesis;
71   end;
72   suppose S1: m = 0;
73     then n = 0;
74::>           *4
75     thus thesis by S1;
76::>              *4
77   end;
78 end;
79
80Corollary:  sqrt 2 is irrational
81  proof
82    assume sqrt 2 is rational;
83    then ex p,q being Integer st
84    q <> 0 & sqrt 2 = p/q;
85::>                     *4
86    then consider m,n being Integer such that
87    A0: sqrt 2 = m/n and m = abs m & n = abs n & n <> 0;
88::>                                                    *4
89    m^2 = 2*n^2;
90::>          *4
91    n = 0 by Lemma;
92::>        *4
93    hence contradiction;
94::>                  *4
95  end;
96
97::> 4: This inference is not accepted
```