Gradual Computerisation and Verification of Mathematics: MathLang's Path into Mizar

Krzysztof Retel

Submitted for the degree of Doctor of Philosophy

Heriot-Watt University

School of Mathematical and Computer Sciences

April 2009

Abstract

There are many *proof checking* tools that allow capturing mathematical knowledge into formal representation. Those *proof systems* allow further automatic verification of the logical correctness of the captured knowledge. However, the process of encoding common mathematical documents in a chosen *proof system* is still labour-intensive and requires comprehensive knowledge of such system. This makes the use of *proof checking* tools inaccessible for ordinary mathematicians. This thesis provides a solution for the computerisation of mathematical documents via a number of gradual steps using the MathLang framework. We express the full process of formalisation into the Mizar *proof checker*.
The first levels of such gradual computerisation path have been developing well before the course of this PhD started.

The whole project, called MathLang, dates back to 2000 when F. Kamareddine and J.B. Wells started expressing their ideas of novel approach for computerising mathematical texts. They mainly aimed at developing a mathematical framework which is flexible enough to connect existing, in many cases different, approaches of computerisation mathematics, which allows various degrees of formalisation (e.g., partial, full formalisation of chosen parts, or full formalisation of the entire document), which is compatible with different mathematical foundations (e.g., type theory, set theory, category theory, etc.) and *proof systems* (e.g., Mizar, Isar, Coq, HOL, Vampire). The first two steps in the gradual formalisation were developed by F. Kamareddine, J.B. Wells and M. Maarek with a small contribution of R. Lamar to the second step. In this thesis we develop the third level of the gradual path, which aims at capturing the rhetorical structure of mathematical documents. We have also integrated further steps of the gradual formalisation, whose final goal is the Mizar system.
We present in this thesis a full path of computerisation and formalisation of mathematical documents into the Mizar *proof checker* using the MathLang framework. The development of this method was driven by the experience of computerising a number of mathematical documents (covering different authoring styles).

To Honorata and Maciej
— my wife and my son —
for their invaluable support and confidence in me

Acknowledgments

As first I wish to thank Fairouz Kamareddine and Joe Wells that they have invited me to their group and supervised me to a great extent for the whole period of my PhD. I especially want to express my gratitude to Fairouz Kamareddine for her constant support on both academic and non-academic life. She helped me find my bearings for life in UK. She was always open and available at any time, whenever I needed help, advice or conversation. I am very grateful to her for building motivation and maintaining higher expectations during my studies and for aiding me in my personal and research development. I want to thank her for the knowledge that I have gained throughout my PhD and that she has passed on to me. Last but not least, I am graciously thankful to her for her confidence in me.

I would like to give special thanks to Andrzej Trybulec who has built the Mizar system and presented it to me during my master studies. I am thankful to him for cultivating my interest in Mizar. I am grateful to him for allowing me to join and to work within the Mizar Group. That he had opened my eyes to the academic environment and he started to build my research development. I would like to express my special gratitude to him for making it possible for me to come and study within the ULTRA group.

I wish to thank Manuel Maarek for his help during the studies. For all those discussions and conversations we held in the office and outside. For being great fellow and friend, and for the knowledge that he has passed on to me.

I want to thank all my colleagues from the Mizar Group for their support, help regarding Mizar and for all that they have taught me.

I would like to thank my lab colleagues, ULTRA members and MACS colleagues for making our workspace comfortable, welcoming and humorous. I wish to thank them for their knowledge sharing and for the time we have spent together.

I would especially like to thank Honorata, my wife and friend, for her continuous help and assistance. I would like to thank her for encouraging and motivating me to complete the work I have started. We were also blessed with our wonderful son Maciej, who has given me happiness and the drive to persevere with the PhD at times where I could not see the end of the tunnel.

Table of Contents

List of Tables

List of Figures

List of Listings

Chapter 1

Introduction

Early mathematics dates back at least to ancient Egypt and Babylonia. This mathematics started as the study of quantity, calculation, measurements and structure and became an integral part of every day life. Over a period of centuries mathematics evolved through abstraction, mathematical logic, logical reasoning, discrete and applied methods to became the mathematics of the world. Today mathematics, seen as "the Queen of the Sciences"[1], is undoubtedly an essential tool in many fields of our lives. It continuously grows with no end sight and as ever, it plays an important role in numerous other disciplines.

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. Mathematicians prefer developing new or studying existing mathematical theories rather than proof checking existing theories on the computer. And so, the gap between the mathematician and the computer proof

[1] as said by Carl Friedrich Gauss

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 proof checking them on the computer. Such computerisations are not sufficiently detailed for correctness verification but are used as *skeletons* in the full formalisation (see F. Wiedijk's work [Wie04a]). 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 [Wie04a], he/she needs to be an expert in Mizar.

Prior to further discussions we want to clarify the difference between two notions of *computerisation* and *formalisation*. These two notions differ in different contexts.

By *computerisation* we mean the process of transforming an informal mathematical document into one of possible computer formats. Such presented document in a computer/programming language can be passed for further manipulation. This manipulation of a document is usually focused on different forms of visualisations. However, from the user point of view, a computerised document can be easily archived, shared or published in different forms.

By *formalisation* we mean the process of translating an informal mathematical document into a formal one. Such translated document can be further passed to a computer software to verify its correctness. In a formalised document, all the ambiguities of the informal text are resolved. Moreover, the fully formalised document contains all possible reasoning/proof steps filled in to the level in which the computer program can verify the document without reporting any errors. The main goal of formalisation is to be able to verify the logical correctness of an informal document with a computer program.

There are various reasons why computerisation should always come before formalisation. First, the computerisation process can be done by the ordinary mathematician without requiring any expert knowledge in programming languages (e.g., computerisation in What You See Is What You Get editor is pretty easy). Secondly, a computerised document can be easily archived and shared amongst other interested people, which is the goal for most mathematicians. Thirdly, the formalisation process is too labour intensive and time consuming and is rarely a goal for mathematicians who usually believe most of their proofs and do not feel they need

to check them by a computer.

The MathLang project aims to give an *alternative* and a *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.

In this thesis, we provide a description on how the ordinary mathematician can do a reasonable amount of work on computerising mathematical texts that will 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 the skeleton of Mizar document 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 into the final Mizar version.

1.1 Motivations

This thesis concentrates on the gradual computerisation of mathematical documents. As the main contribution to the MathLang project, we developed the Document Rhetorical aspect that allows capturing and annotating the narrative structure of mathematical documents.

The list below provides our main motivations to the research we carried out:

1. *To handle the structure of a mathematical document as it appears on paper and at the same time to allow further computerisation and analysis.* Our proposed annotation system can deal with different styles of writing mathematics.

2. *To allow the presentation of a text with different layouts.* Currently the presentation of the structure of a documents is rather linear and it is not clear which parts (chunks of text) of the document depend on which (which theorem depends on which lemma or definition etc.). Ideally the presentation of a document should be flexible, and should allow the full automatic generation of different views of the structure of a document. For this reason, we introduce in this thesis new notions like: *Dependency Graph*, *Graph of Logical Precedences*, skeleton of the document in a chosen formal system, etc.

3. *To allow further formalisation.* Capturing the narrative structure of a document is not only for computerisation purposes, but also for further formalisation. The automatically generated views of the narrative structure of a text are very important to generate further forms of the text including a more formalised version (in a chosen formal system). We concentrate, in this thesis, on presenting mathematical documents in the Mizar language.

1.2 Contributions

We summarise the contributions of this thesis in the following list:

- MathLang's Document Rhetorical aspect (DRa). This aspect captures the narrative structure of the mathematical document. It consists of:
  - A formal abstract syntax based on well-known standards,
  - A DRa ontology and a related annotation system,
  - A set of encodings of mathematical documents as a test case of the aspect's expressiveness,
  - A set of rules for DRa annotations,
  - An implementation of annotation rules within a user-friendly scientific editor $\mathrm{TEX_{MACS}}$,
  - An implementation of XSLT files for automatically transforming DRa encoded documents into graph presentations,
  - A set of different graph presentations of a particular annotation.
- A gradual computerisation path into Mizar. This is the first full computerisation path starting from a CML document and applying MathLang aspects and tools to reach different versions of the document including its Mizar FPS version and ending in its full formalisation in Mizar. This consists of:
  - A clear, concise transformation path from a literary document into more formal representations (i.e., different MathLang's aspects, and Mizar FPS) and finally into the fully formal document (i.e., fully formalised in Mizar),
  - A set of transformation hints from the annotated narrative aspect of a document into the Mizar document skeleton,

– A comparison between on one hand, the MathLang document preamble and MathLang constructs, and on the other hand, the Mizar document environment and Mizar symbols and constructions.

– A number of examples as a test case of gradual computerisation.

These contributions have been published in a number of articles. All of These publications are co-authored with F. Kamareddine, J.B. Wells and M. Maarek. The content of these publications is presented and spread across the chapters of this thesis.

One article [KMRW07b] was published in the proceedings[2] of the MKM 2007 conference – the Sixth International Conference on Mathematical Knowledge Management. At this conference K. Retel gave a talk presenting his work on the DRa aspect of the MathLang project.

Another article [KMRW07a] was published[3] in a book entitled "From Insight to Proof, Festschrift in honour of Andrzej Trybulec". This book was published to honour the 65th anniversary of A. Trybulec.

Another article [KMRW07c] was published in the journal "Review of the National Center For Digitization", published by the Faculty of Mathematics at the University of Belgrade. This article is a short description of the MathLang project.

The work presented in these publications and this thesis was extensively discussed during weekly meetings of the MathLang group. Moreover, this work has gained from comments, inputs and collaborations with members of the MathLang project F. Kamareddine, J.B. Wells, M. Maarek and P. van Tilburg. The work has been presented in a number of talks[4] given by K. Retel. These talks include: a presentation at the MKM 2007 conference, a presentation at the Ωmega+MathLang workshop held by Ωmega group at the University of Saarlandes, and a number of presentations at PhD seminars series at Heriot-Watt University.

1.3 Outline

In Chapter 2, we begin with a description of the computerisation of mathematics. First we describe work that has been carried out in this field. Then we present

[2]The proceedings were published by Springer-Verlag in a series of Lecture Notes in Artificial Intelligence

[3]A book was published by the University of Bialystok, as a part of the special series of *Studies in Logic, Grammar and Rhetoric* journal.

[4]Presentation slides and materials are available at `http://www.macs.hw.ac.uk/~retel/`

in more details the MathLang project, its goals, philosophy and origin. We also discuss briefly MathLang's design approach and its current development.

In Chapter 3, we reflect on the Mizar system. We start by presenting a general description of the Mizar language and of Mizar documents. This general discription is illustratd by an example. Moreover, we report on the Mizar Mathematical Library (MML) – the biggest library of computer verified mathematics around the world. In the process we describe the MML Query language and discuss the types used in Mizar. We also report on a number of formalisations that the author of this thesis performed. Furthermore, we describe briefly the use of the Mizar system as a tool for teaching mathematics. The author of this thesis was involved in a number of courses that were taught at the University of Bialystok. Finally, we present the Mizar FPS, a representation of mathematical documents in the Mizar language which explicits the content of a document in the Mizar language but does not fill all the reasoning holes.

Chapter 4 reflects deeply on our main contribution to the MathLang project, the Document Rhetorical aspect (DRa). First, we present an overview of existing formats and we defend the need of the DRa in MathLang. Then we move to describe the ontology of this aspect and to demonstrate the annotation process with an example. We report on the plan and concrete syntax of the markup language for the DRa. We also discuss different representations by means of graphs that are built automatically from MathLang annotated document. We define our contribution in the form of a Dependency Graph and a Graph of Logical Precedences. Finally, we provide a short description of our contribution to the automatic analysis of these graphs.

In Chapter 5, we present our approach and contribution to the computerisation mathematics. First, we draw a picture of possible formalisation paths from a CML document to Mizar. Thereafter, we demonstrate with an example, the computerisation process of a mathematical document which follows the path from CML to full Mizar passing through the various stages obtained by the MathLang aspects and Mizar FPS. We explain in details how the narrative features can help to build a skeleton of a document in the Mizar language. We provide a number of hints that could be expanded to build a computer software that would assist the user during the formalisation process. Finally, we compare the MathLang preamble of an annotated document to the Mizar environment. We also provide a comparison between MathLang constructs and their Mizar counterparts.

Chapter 6 describes the implementation of the DRa aspect. We present the

$\TeX_{\text{MACS}}$ side of the implementation. We also show the XSLT document that allows the automatic transformation of the MathLang-DRa annotated document into its graph presentation. Finally, we describe an informal algorithm for checking the well-formation and annotation of a document with the DRa markup language.

Finishing the thesis, Chapter 7 describes current developments and related work before concluding. First, we report on the current development status. Next, we describe further challenges and aspects that the MathLang group envisions. We also provide a short discussion regarding the DRa and related works. In particular we shortly compare the DRa markup language with OMDoc. Finally, we conclude this thesis.

Chapter 2

MathLang and Its Aspect Oriented Design

In this chapter we present existing tools for computerising and formalising mathematics. We give an overview of existing *proof systems*. We also discuss different approaches to the formalisation of mathematics.
The main part of the chapter presents the MathLang project, its design and development. We explain the experience driven development of the project which lead to its aspect oriented design. We give an overall view of the MathLang framework. We also present MathLang encoding facilities via a number of short mathematical examples.

It is highly important to note that this chapter is a survey chapter. This means that the content of this chapter might contain a direct or indirect relations, citations, examples and express similar or very close ideas that were written by members of MathLang group in a number of existing papers, reports, talks and M. Maarek PhD thesis. Since it is survey chapter we heavily use materials from [KW01a, KW02, KMW04b, KMW04a, KMW06, Ret05a, KLMW07, KMRW07b, KMRW07c, KW08, KWZ08].

2.1 Computerisation of Mathematics

2.1.1 The Commmon Mathematical Language

Mathematicians use a distinctive language style as a communication medium. This is the same language, which has been used for many years to represent mathematics, and which we are tought from our early years at school to the university level.

It allows to represent equations, geometry or any problem of mathematical nature. This informal every-day mathematical language has been called Common Mathematical Language (CML). The CML texts are a mixture of symbols composed into formulas and natural language chunks like nouns, adjectives, verbs and sentences. For the benefit of this chapter, toy examples of CML texts are presented in Figure 2.1 and Figure 2.2. We will use both examples to present different representation of the same text through different systems.

Proposition 1. *For any two sets A, B, holds $A \subseteq B \Longrightarrow A \backslash B = \emptyset$.*

Proof. We prove the theorem by contradiction. Suppose $A \subseteq B$ and $A \backslash B \neq \emptyset$. Then exists element $x \in A \backslash B$, which implies that $x \in A$ and $\neg(x \in B)$. Since $A \subseteq B$, we know that every element of A is an element of B. In particular this holds for x and therefore $x \in B$, which contradicts our assumption that $x \notin B$. □

Figure 2.1: A set theory proposition and its CML proof.

The past forty years have seen a sharp increase in the use of computers by mathematicians for their work purposes. Such use covers communication, authoring, process and checking/verifying mathematical knowledge. There exists already a number of flexible computer tools that can represent mathematical knowledge in various ways.

2.1.2 Typesetting Systems

Typesetting systems like LaTeX, TeXmacs or open office-suits are widely used by mathematicians and anyone who can produce a document for viewing and/or printing. Their document format is usually used for archiving and storing.

Theorem 43 (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 $(a, b) = 1$. □

Figure 2.2: Pythagoras' proof of the irrationality of $\sqrt{2}$ by G.H. Hardy and E.M. Wright [HW80] as presented by F. Wiedijk in [Wie06]

LaTeX is the most commonly used system within the academia environment due to the typesetting quality and automation, as well as a high visual appearance. It is widely accepted by mathematicians and publishers of mathematical journals and books. LaTeX is a system and a programming language that provides the required expressiveness over the structure of a document and its presentation layer. It supports any document structure to the extent of correct visual appearance.
A mathematical text written within LaTeX is well structured and is a computerised version of a CML document. The logical structure of symbolic formulas is not directly represented. Moreover, the understanding of the structure and mathematics represented within the document is left for the reader and requires some degree of mathematical knowledge from the reader. The mixture of formula level with natural language certainly helps the human reader to understand the text, but the structure of the document depends on the author's style of writing which can make the understanding process more complex. This makes it even more difficult for computer programs to automatically recognize semantics. The current stage of the automated recognition of the semantics of natural language text is not yet in use within practical systems, due to its poor representation of mathematical meanings. As a consequence there is no computer software which allows to check and verify the logical correctness of mathematics represented this way. Moreover there is no semantic search engines that parses such created mathematical documents.

```
\begin{proposition}
  For any two sets $A,B$, holds
  $A\subseteq B \implies  A\backslash B =  \O$.
\end{proposition}
\begin{proof}
  We prove the theorem by contradiction. Suppose
  $A\subseteq B$ and $A\backslash B \neq \O$. Then exists
  element  $x\in A\backslash B$, which implies that $x\in A$
  and $\neg(x\in B)$. Since $\subseteq B$, we know that every
  element of $A$ is an element of $B$. In particular this
  holds for $x$ and therefore $x\in B$, which contradicts our
  assumption that $x\notin B$.
\end{proof}
```

Listing 2.1: The LaTeX code corresponding to the example of Figure 2.1

TEXMACS [vdH01, vdH04] is a free scientific text editor which offers the same typesetting quality as LATEX. The goal of TEXMACS is to provide a What You See Is What You Get (WYSIWYG) editor that still makes it possible to write correctly structured documents and handle mathematical formulas with aesthetically pleasing typesetting results. TEXMACS is not a front-end for LATEX but at the end can produce a document in TEX and LATEX-like format. The internal representation of a TEXMACS document is a mixture of XML-like tree structures and LATEX-like commands, see Listing 2.2 for an example with the source code of a document shown on Figure 2.3.

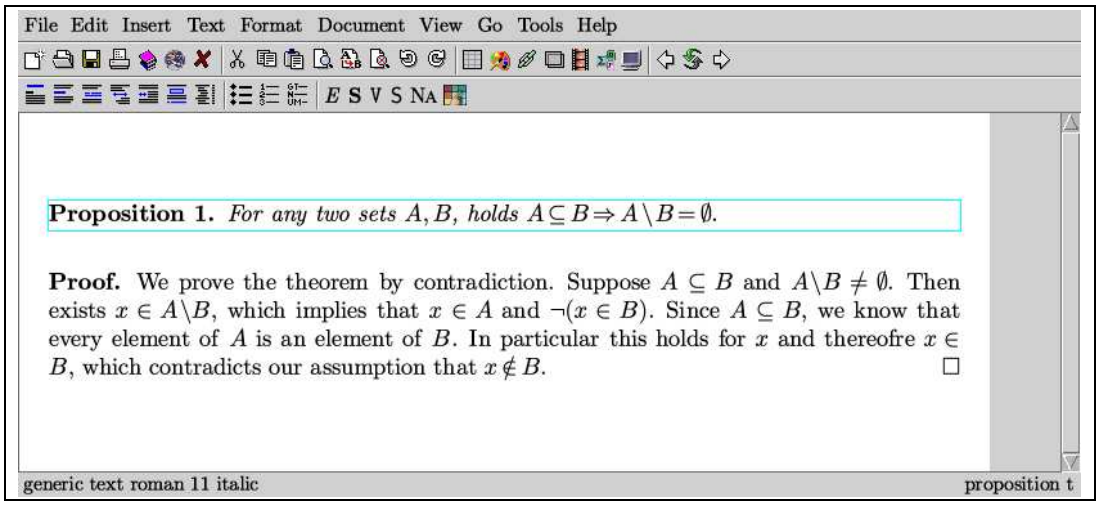


Figure 2.3: The TEXMACS user interface

Compared to LATEX or TEX, TEXMACS is used as a front-end for a number of external systems like computer algebra systemCASs or theorem provers. This is possible due to the TEXMACS plugin system. There exists already a number of TEXMACS interfaces for systems like Coq (TmCoq and tmEgg which are respectively presented in [AR03] and [MG06]) and systems like the Ωmega proof assistant (presented in [ABFL05, WAB06, ABFL06, AFNW07].
Due to the TEXMACS capabilities and features, the MathLang group has decided to use TEXMACS as an interface to the MathLang framework. We discuss in details our use of TEXMACS in Section 5.2.1.

```
<TeXmacs|1.0.6.11>

<style|generic>

<\body>
  <\proposition>
    For any two sets <math|A, B>, holds <math|A\<subseteq\>B \<Rightarrow\> A
    \<backslash\> B = \<emptyset\>.>
```

```
  </proposition>

  <\proof>
    We prove the theorem by contradiction. Suppose <math|A \<subseteq\> B>
    and <math|A\<backslash\>B \<neq\> \<emptyset\>>. Then exists <math|x
    \<in\>A\<backslash\>B>, which implies that <math|x\<in\>A> and
    <math|\<neg\>(x\<in\>B)>. Since <math|A\<subseteq\>B>, we know that every
    element of <math|A> is an element of <math|B>. In particular this holds
    for <math|x> and thereofre <math|x\<in\>B>, which contradicts our
    assumption that <math|x\<notin\>B>.
  </proof>
</body>

<\initial>
  <\collection>
    <associate|font-base-size|11>
    <associate|language|british>
  </collection>
</initial>
```

Listing 2.2: TEXMACS sources of the document shown in Figure 2.3

2.1.3 Markup Languages

Quite recently markup languages became very popular. One of the reasons is mainly the features of a markup language regarding structuring a document in a format that computers can easily parse, maintain and process further. Furthermore, a document annotated with a markup language is standardised. The main format that is currently in use is called Extensible Markup Language (XML). It is a format of choice for the future millennium. It is called "extensible" because it allows its users to define their own elements which in meritum it allows the creation of custom markup languages.

XML was one of the driving purposes for the development of mathematical markup languages. These mathematical markup languages were originally aimed at offering a standardised and open format for encoding mathematical formulas. Currently there exists a number of such languages and systems that allow to write mathematics in a more semantically oriented document representation like OpenMath, MathML and OMDoc.

These systems are better than the typesetting systems at representing the knowledge in a computer-accessible way. There exists support for converting such created documents into their representation in typesetting systems, although, in practise it is still difficult and labor-intensive to have a full control over the visual presentation.

Although markup languages are better for capturing the semantical structure of a mathematical document, type checking symbolic formulas is still not handled by these systems. Moreover their usage is far much more time expensive and tedious than the usage of typesetting system. Therefore practicing mathematicians avoid and object to the use of markup languages as a daily tool for authoring mathematics. However, to be more precise it would be wrong to say that mathematicians do not use them at all. Mathematicians use OMDoc, for instance, for a representation of their document into a more standardised format and as a step toward further semi-automatic manipulation and transformation of the document.

Mathematical documents written/annotated using markup languages can follow some validation rules for checking their well-structured format. Usually markup languages provide a standard well-formedness validation. Due to the fact that mathematical markup languages come from the same family of the XML language, they do follow a standard XML well-formedness validation, i.e. each tag has to be closed properly. This validation type is similar to the one that the LaTeX system offers while processing and generating a visual representation of a document.
The advantage of the validation of mathematical markup languages is a validation of some semantics rules that are usually created by the user (who developed the markup language) or are expressed within the XML schema or the DTD.

There exists also a prototype of validation tool for the OpenMath objects which is simply a syntactical analysis of the structure of the XML encoding of the OpenMath object. A semantical validation process for OpenMath objects is described in [CC99] by O. Caprotti and A. Cohen. It uses the Extended Calculus of Construction (ECC) to validate OpenMath contents. In [Dav99], a more complex system, the Simple Type System (SST) [Dav00], has been used for the same purpose. But unfortunately, no implementation of these validation processes are currently available.

```
<assertion id="th" type="theorem">
  <commonname> Pythagoras' Theorem
   <FMP> <OMOBJ> √2 ∉ ℚ
   <CMP> <OMOBJ> √2 is irrational.
<proof id="pr-th" for="th">
   <CMP> If <OMOBJ> √2 is rational, then the equation
         <OMOBJ> a² = 2b² is soluble in integers <OMOBJ> a,
         <OMOBJ> b with <OMOBJ> (a, b) = 1. Hence <OMOBJ> a² is
         even, and therefore <OMOBJ> a is even. If
         <OMOBJ> a = 2c, then <OMOBJ> 4c² = 2b², <OMOBJ> 2c² = b²,
```

```
and <OMOBJ> b is also even, contrary to the
hypothesis that <OMOBJ> (a, b) = 1.
```

Listing 2.3: An OMDoc/OpenMath encoding of the example of Figure 2.2.
For readability and brevity, we show only the opening tag of each XML element; instead we use indentation to express nesting. We also use traditional mathematical output prefixed with the `OMOBJ` *tag for* OpenMath *formulas instead of showing the XML tree.*

2.1.4 Proof Systems

There are formal *proof systems* that could be divided into two groups: (1) *proof assistants* (sometimes called *proof checkers* like Mizar[1] [Try80, Rud92, MR05], Isabelle[2] [NPW02], Coq[3] [Log06, BC04], etc.), and (2) *theorem provers* (like Boyer-Moore[4] [KB95], PVS[5] [ORS92], Vampire[6] [Vor95, RV02], etc.). Each proof system provides its own formal language for writing mathematics based on some foundation of logic and mathematics.

The very first project of computer support for formal mathematics representation and automated verification dates back to 1967. At that time N.G. de Bruijn initiated an Automath[dB80, vD80] (AUTOmating MATHematics) project aimed at designing a language for expressing complete mathematical theories in such a way that a computer can verify the correctness[vBJ77]. Automath supported automated checking of full correctness of a mathematical document written in Automath's formal language. Six years later around 1973-74, independently and without the knowledge of the existing Automath project, A. Trybulec has started a project called Mizar. The vision was to develop a computerised assistance in the process of editing mathematical papers. The main aims of the early years of that project were:

1. to form a basis for the construction of an automated information system for mathematics,

2. to facilitate the detection of errors, the verification of references, the elimination of repeated theories, etc.,

[1] `http://www.mizar.org`
[2] `http://isabelle.in.tum.de`
[3] `http://coq.inria.fr`
[4] `http://www.cs.utexas.edu/users/moore/best-ideas/nqthm/`
[5] `http://pvs.csl.sri.com/`
[6] `http://www.voronkov.com/vampire.cgi`

3. to open a way to a machine assisted education of the art of proving theories,

4. to enable the automated generation of input into typesetting systems.

We discuss in more details the Mizar system in Section 3.

Since then, many *proof systems* have been built to mechanically check logic, mathematics or software (e.g., Coq, Nuprl, Isabelle, Otter etc.). Generally these systems support checking full correctness of mathematical theories.

Unfortunately, along with the number advantages of using *proof systems* there are number of disadvantages. The main disadvantage is the enormous expense of formalisation using any of the existing *proof systems*. Let us explain briefly what causes this issue. First of all a mathematician wanting to build a formalisation needs to decide which *proof system* he wants to use for writing and verifying the logic of a document. At this point he is obliged to build a formal document within the language and the logic foundation supported by the chosen software. This is already a very labor intensive job. First of all, because it requires an expert knowledge to the underlined logic and the syntax of the language interpreted by the chosen *proof system*. Second of all, most proof systems have no meaningful support for the common mathematical language. In most cases the language used to build formalisation for a specific *proof system* differs greatly from CML. The rigid structure and rules of a formal document created within any *proof systems* make the language of the formal system much more closer to a computer programming language than to a natural language. In many cases it has nothing common with a natural language. A notable exception is Mizar, which however requires the use of natural language in a rigid and inflexible way. The Mizar formalised version of a mathematical document is human readable and resembles the CML, although it still looks like a computer programming language.

Listings 2.4 and 2.5 present the formalisation of a CML document from Figure 2.2 within two *formal systems*: Mizar and Coq, respectively.

```
theorem :: Phytagoras' theorem
  sqrt 2 is irrational
proof
  assume sqrt 2 is rational;
  then consider a,b being Integer such that
  A1: b <> 0 and
  A2: sqrt 2 = a/b and
  A3: a gcd b = 1 by Local_TH2;
  A4: b^2 <> 0 by A1,SQUARE_1:73;
  0 <= 2 by NAT_1:18; then
```

```
  2 = (a/b)^2 by A2,SQUARE_1:def 4
       .= a^2/b^2 by SQUARE_1:69;
  then A6: a^2 = 2*b^2 by A4,XCMPLX_1:88;
  then a^2 is even by ABIAN:def 1;
  then a is even by PYTHTRIP:2;
  then consider c being Integer such that
  A8: a = 2*c by ABIAN:def 1;
  A9: 4*c^2 = (2*2)*c^2
            .= 2^2*c^2 by SQUARE_1:def 3
            .= 2*b^2 by A8,SQUARE_1:68,A6;
  2*(2*c^2) = (2*2)*c^2
            .= 2*b^2 by A9;
  then 2*c^2 = b^2 by XCMPLX_1:5;
  then b^2 is even by ABIAN:def 1;
  then b is even by PYTHTRIP:2;
  then ex j being Integer st b = 2*j by ABIAN:def 1;
  then 2 divides a & 2 divides b by A8,INT_1:def 9;
  then A11: 2 divides a gcd b by INT_2:33;
  a gcd b = 1 by A3,INT_2:def 4;
  hence contradiction by A11,INT_2:17;
end;
```

Listing 2.4: A part of a Mizar proof of the example of Figure 2.2
The Mizar version of the proof, inspired by F. Wiedijk's one from [Wie06].

All in all, a formalisation process using any of the *proof systems*, in smaller or bigger level, is similar/could be compared with writing a computer program where a *proof system* language corresponds to a "programming language" and the *proof system* itself, plays the role of a "compiler". As a result of these disadvantages, mathematicians are required to possess some computer programming knowledge. This extended knowledge is too domain specific and totally unnecessary for the ordinary mathematician. Therefore, ordinary mathematicians are usually not willing to spend the effort to formalise their own work in a proof system.

Going farther, mathematics written on a piece of paper has many of "justification holes" and "hints" which are left for the reader's intuition. Mathematicians assume that some of their justifications are obvious or require a very little effort or depend on straightforward reasoning. Hence, they tend not to include obvious proofs in their papers and at the same time make the paper clear and concise in its structure and logical reasoning. This of course depends on an individual mathematicians style of writing mathematics. When mathematics is written in a formal language of a chosen *proof system*, those "justification holes" and "hints", as well

as any obvious inferences that are left implicit , have to be made explicit within a formal document. *Proof systems* have no "human mind intelligence" to understand those implicit justifications while checking the logical correctness of a document. Therefore any implicit information in a document needs to be made explicit within a formalised representation of the document within a chosen *proof system*.

A *proof system* can check the logical correctness of a document only if it is sufficiently detailed for the software. As a result, the formalisation of mathematics becomes even more labour intensive and tedious for a mathematician. Moreover the formalisation of a document is not as clear or as concise as the original paper-printed version of the mathematical document.

```
Theorem irrationalRsqrt2: (irrational (sqrt (S (S O)))).
Red.
Intros p q H; Red; Intros H0; Case H.
Apply (main_thm p).
Replace (Div2.double (mult q q)) with (mult (S (S O)) (mult q q));
 [Idtac | Unfold Div2.double; Ring].
Case (Peano_dec.eq_nat_dec (mult p p) (mult (S (S O)) (mult q q))); Auto;
 Intros H1.
Case (not_nm_INR ? ? H1); Repeat Rewrite mult_INR.
Rewrite <- (sqrt_def (INR (S (S O)))); Auto with real.
Rewrite Rabsolu_right in H0; Auto with real.
Rewrite H0; Auto with real.
Cut ~ <R> q == R0; [Intros H2; Field | Idtac]; Auto with real.
Apply Rle_ge; Apply Rlt_le; Apply sqrt_lt_R0; Auto with real.
Qed.
```

Listing 2.5: A part of a Coq proof of the example of Figure 2.2. The above code in Coq is a "declarative approach" of the presentation of the CML document. *Part of Laurent Théry's Coq proof from F. Wiedijk's [Wie06].*

2.1.5 Other systems

There exist a number of other systems that are widely used for representing mathematics on computer. Computer Algebra SystemCASs are one of them. These software environments, like Maxima[7], Maple[8], Mathematica[9], Matlab[10] etc., are designed for facilitating symbolic and numeric computations and are mainly used in the fields of mathematics where the symbolic computations are the main area of investigation.

[7] http://maxima.sourceforge.net/
[8] http://www.maplesoft.com/products/Maple11/
[9] http://www.wolfram.com/products/mathematica/
[10] http://www.mathworks.com/products/matlab/

Each CAS has its own language for representing expressions, mathematical statements and describing computations. The CAS language has almost no support for embedding natural language neither for precise control over typesetting. Moreover, computer algebra systemCASs do not support any form of checking logical correctness of mathematical statements. However, a CAS document could be converted to other formats and then passed to another software for logical verification. There are also a number of advantages of the usage of CASs. For instance, CAS handles the computation and representation of matrices whereas *proof systems* do not have such support. You can find a way to express matrices in *proof systems* but the visual presentation of such matrices is not as clear as and suffers greatly from the presentation found in books.

Another approach to computerising mathematics requires the scanning of images of pages of mathematical papers using Optical Character Recognitions (OCR) techniques. There exists a number of OCR programs that capture a structure layer of a mathematical document and make the document search-able. However, high quality OCR for capturing the full semantic of a mathematical documents is still an area with significant research challenges. Moreover, all OCR systems work on already digitised documents and do not support any form of authoring.

2.2 The MathLang Project

MathLang is a framework for mathematics on computers.

1. *MathLang is* ***a framework****.* It is meant to be used for communication and as a concrete support for human mind formulation. MathLang is a well structured framework aimed to synthesize the common mathematical language.

2. *MathLang is* ***for mathematics****.* It is meant to be open to any branch of mathematics and to any topic that uses mathematics as a base language. MathLang mimics mathematics in its incremental construction of a body of knowledge.

3. *MathLang is* ***for computerisation****.* MathLang is meant to be a medium for a human-system, human-human via a digital support, and system-system communication. MathLang is a computer-based framework and therefore offers automation facilities.

(from M. Maarek's PhD thesis [Maa07])

2.2.1 MathLang's Goals and Philosophy

The MathLang project dates back to 2000 when F. Kamareddine and J.B. Wells started the project within the ULTRA (Useful Logics, Types, Rewriting and Application) group. In 2000 and 2001 in the MathLang's proposals [KW00, KW01a], they expressed the idea of developing a new mathematical language and framework called MathLang[11], which keeps most of the advantages of CML and avoids its disadvantages ([KW01b, KW02, KW08, KWZ08]). Moreover the idea was to provide a *second language* to CML, so any mathematical document written in CML could be written instead in MathLang, which would allow a gradual computerisation and formalisation of mathematical texts. The initial MathLang project has fine and solid goals:

1. A MathLang text written in a formal language MathLang should remain very close to CML and should support all the usual features of the CML: natural language notions, document structure, symbolic formulas, images. In such case, MathLang can be used as a *second language* to the CML.

2. It should be possible to write a MathLang text in a less ambiguous way than the corresponding origin text. A MathLang document should support representing the semantics and structure of the original document. The support for semantics should cover formula level as well as the entire document and its relationship to other documents, so it supports building computerised and connected libraries of mathematics.

3. The structure of a MathLang document has to express exactly the structure of a CML text, so the reading and writing of a MathLang text should be close to that of reading and writing CML. At the same time the MathLang structure has to be more precise and less ambiguous than that of CML documents. MathLang should somehow support and help mathematicians to precisely identify the logical structure of a document, without requiring readers and authors to adapt their thinking to fit the formal representation of MathLang.

4. A MathLang language does not restrict mathematicians to any existing theoretic foundations (like set/type/category theory). So it allows to capture all kind of mathematics and provides a freedom for the mathematician to de-

[11] The project was always named MathLang but the initial name of the proposed language was NML (New Mathematical Language).

velop any mathematical document regardless of the branch of mathematics or even the level of correctness.

5. The authoring of a MathLang text should not require "extra" knowledge, skills or significant effort from the author. More specifically, it should not be more difficult than authoring mathematical documents using typesetting systems like LaTeX.
   Furthermore post-authoring additional features (such as full formalisation in a proof system) should also not be too labor-extensive for the author.

6. A MathLang text may act as a communication medium for the human and the reasoning, expressed in CML is computerised and ready for further manipulation and computer-based analysis. MathLang's document uniformity provides a firm basis for communication allowing many people to work productively with the same text. As MathLang documents are more precisely structured than CML texts, they provide an excellent ground to the administration of mathematical knowledge.

7. The MathLang framework provides extra features supporting more rigor to the initial translation of CML. One can define further levels of translations into more semantically and logically complete versions. Each level is a refined version of the previous level and captures/annotates more logics and semantics of the original document. This gradual computerisation method should be more accessible than direct formalisation, because a number of first levels do not require any particular expertise in formalisation.

8. The MathLang document structure should allow and support further post-authoring computer manipulations that respect the mathematical structure and meaning. Examples include the translation of the MathLang document into different format (e.g., LaTeX, OMDoc), the high-quality visual representation (similar to the one that LaTeX provides), semantic-based searches, the extraction of proof skeletons and sketches, etc. In particular it should support interfacing with proof systems so that parts of the MathLang document can contain full formal details in a chosen foundation and those parts can be automatically verified by the proof system.

None of the previous computer softwares for representing mathematical documents satisfies the goals of MathLang. Therefore the MathLang is not yet another framework for computerising mathematics but it is an approach to bridge the gap

between mathematicians writing day-to-day CML texts and the proof systems community. Moreover, MathLang's philosophy is to bridge the gap between CML and full formalisation.

As expressed above, the main goal of MathLang is to support different degrees of formalisation. Furthermore, for those mathematicians where the full formalisation of an authored document is a goal, MathLang is intended to achieve this by providing a framework where the full formalisation can be accomplished in gradual steps and at the same time the expertise knowledge of a chosen proof system is postponed to the latest phases of the formalisation. Full formalisation is sometimes desirable but also is often undesirable due to its labour and time expense as well as the requirement to commit to many inessential details of document to be formalised. Therefore MathLang could support partial formalisation or full formalisation of chosen parts of a MathLang document. The formalisation degree completeness is left for the author to decide how far he wants to go. Moreover, a partially formalised document can be tackled later on by another person.

Summarising, the novel approach for putting mathematics on computer proposed by MathLang desires to follow three main goals:

- Providing an authoring framework for mathematicians that is very similar to the one they are used to work with, where the front end language of that framework mimics and contains all features of CML. This allows mathematicians to work as they like using their own style of writing mathematics. At the same time, the framework gives the author a freedom to develop any mathematical document regardless of the branch of mathematics or even the level of correctness.

- Bridging the gap between CML and full formalisation. This opens the possibility of further (sometimes post-authoring) semantic and logic refinements of a computerised text to reach the desired formalisation level. This should support either a partial formalisation or a full formalisation of a whole document or simply some required parts of a document whichever is desired. This allows to postpone, the expertise knowledge of a chosen proof system needed to fulfill the essential details to meet the formalisation level allowing computer verification of logical correctness.

- Providing a compatible way with future extensions of the framework to support additional uses of mathematical knowledge. The design of MathLang language should provide a bridge to existing languages like OMDoc,

TEXMACS.

2.2.2 MathLang's Origin and Design Approach

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 [dB87] (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 elemenary 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. *[dB91]*

MV was proposed as a formal substitute for parts of CML. R. Nederpelt refined MV into another formal substitute for parts of CML, Weak Type Theory (WTT) whose underlying proof theory was developed by Kamareddine [NK01, KN04]. MathLang started from de Bruijn's path idea and took R. Nederpelt's WTT as the initial language at the lowest level in the path. Soon, MathLang 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. 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 process of designing MathLang is handled during a number of gradual phases. This iterative process is shown in Figure 2.4. The main development is based on encoding an existing mathematical text within the MathLang framework. On each iteration the MathLang framework evaluates a translation of a mathematical document, during which difficulties may be experienced. This leads to

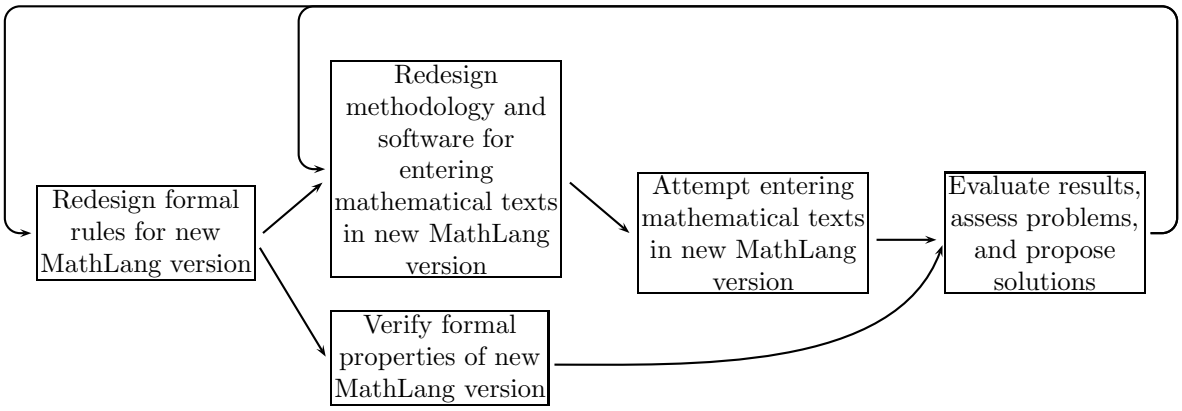


Figure 2.4: The MathLang development process, presented in [KW02, KW08, KWZ08].

determine new solutions for the encountered difficulties and further design adjustments in MathLang. As a result this may affect the need to redesign formal rules for representing mathematics in MathLang, as well as, to redesign the methodology for capturing texts in this representation, and supporting software.

The mathematical texts chosen for the design purposes cover a wide spectrum of mathematics. Our choice is oriented toward the variety of writings by ordinary mathematicians.

We take into account a number of factors before we chose a mathematical text for encoding purposes. Among those factors are the following:

- represent the variety of mathematical writings by ordinary mathematicians rather than mathematical logicians or set-theorists,
- capture different styles of writing mathematics, as well as different mathematical theories, (e.g., number theory, topology, algebraic geometry etc.),
- encode modern and historical mathematics,
- choose texts that have been previously formalised by others.

Extensive computerisations of different mathematical texts (some taken fully from natural language to different levels of computerisation and finally to full Mizar), continue to shape the MathLang language. Its expressiveness has been increased in comparison to MV and WTT.

The elements of the formal syntax of WTT, heavily based on MV, are classified to four linguistic/grammatical categories/levels:

1. *Atomic* level: *variables, constants* and *binders*,

2. *Phrase* level: *terms, sets, nouns* and *adjectives*,

3. *Sentence* level: *statements* and *definitions*,

4. *Discourse* level: *context, lines* and *books*.

In the terminology of WTT, a document is a *book* which is a sequence of *lines*. Each line can be decomposed as a pair of: a *context* and a *sentence*, whereas a *context* contains facts assumed or declared in a *sentence*. WTT has three different ways of introducing names: *definition*, *declaration* and *context*. There is a subtle difference among all three ways of introducing names. It is related to the scope (life time) of the introduced name within the document. *Definition* introduces a name whose scope is the part of the document following the definition and is limited to the closure tag of that part of the document where it was introduced. Furthermore, *definition* can have parameters whose scope is the body of the definition.
A *declaration* in a context introduces a name (without parameters) whose scope is only the current line. Another difference between a *definition* and a *declaration* is that *definition* defines a new symbol in mathematical texts, whereas a *declaration* introduces a new symbol primarily without representing a meaning of that symbol. Finally, a *preface* for a *book* introduces a number of names whose scope is limited to the document. Compared to the definitions, names introduced in the *preface* are *constants* (having parameters and ranges as *weak types*) whose meanings are not provided.
Declarations, definitions and statements can consists of *phrases* which are built from *terms, sets, nouns* and *adjectives*.

WTT uses a weak type system to verify and check the well-formedness of a document written/formalised in WTT. By extension a word *weak*, associated to the phrase *type system*, indicates that the typing system is light and provides a generic judgement based on those types. WTT defines eight types - `book`, `context`, `statement`, `definition`, `term`, `set`, `noun` and `adjective` - which are directly related to the grammatical categories of the abstract syntax of WTT.

WTT was considered as a *mathematical vernacular* for mathematicians and provides a lot of useful ideas. Although, its definition has a number of limitations. For instance, WTT does not provide nor support a way to reuse a theory or a concept introduced in one document within another document. This builds difficulties during the process of creating a digital library of mathematical documents written

in WTT.
WTT does not provide a clear indication/annotation of which statements are used as a reasoning to other statements. In general WTT does not deal with proofs and logical correctness.
Moreover, WTT has a lack of annotation of a mathematical roles (e.g., *proof, theorem, lemma*, etc.,) for a group o statements. Furthermore, it does not allow to associate human readable labels for specific groups of statements (e.g., "Pythagoras Theorem").
There is no support for embedding "natural" language within the WTT formalised document. This causes the WTT formalised document to be "user unfriendly" and awkward to read.

Despite all those limitations WTT introduces the best approaches and solutions taken from MV. This was revised and provided with more clear and precise concepts, that evolves into a machine readable language with a weak type system. We used WTT development in the initial stages to shape the path from the common mathematical document to its fully formal representation verifiable by a computer software.

2.2.3 The Current MathLang Design

MathLang adopted the decomposition of the computerisation process by a number of *levels* (see section 2.2.2). The notion of *level* (used in the early stages of the MathLang project) implies an obligation to meet one level before another. This provides a stratification of the formalisation process. Following this stratification a large number of students, including short projects (by either 4th year undergraduate students: H. A. Ross, M. I. Lopez Fernandez, M. Petrie, A. Brand, or M.Sc. students: A. Retzepi, A. Tsaousis, Jing He and A. Asimakopoulos) and Ph.D. studies (by 4 students: M. Maarek, K. Retel, R. Lamar and C. Zengler) have carried out a number of research and experiments on the various computerisation in MathLang framework. As a result, J.B. Wells proposed in 2005 replacing the *levels* of formalisations by the so-called **aspects**. The notion of *aspect* permits a greater focus on the knowledge captured by each decomposition element. The notion of the *aspect* also prevents the misunderstanding of the process of computerisation of mathematical documents using the MathLang framework. Each recognised aspect could be done sequentially, simultaneously or independently without a requirement of meeting the prior *aspect* computerisation.

In the current development of MathLang we have identified, designed, for-

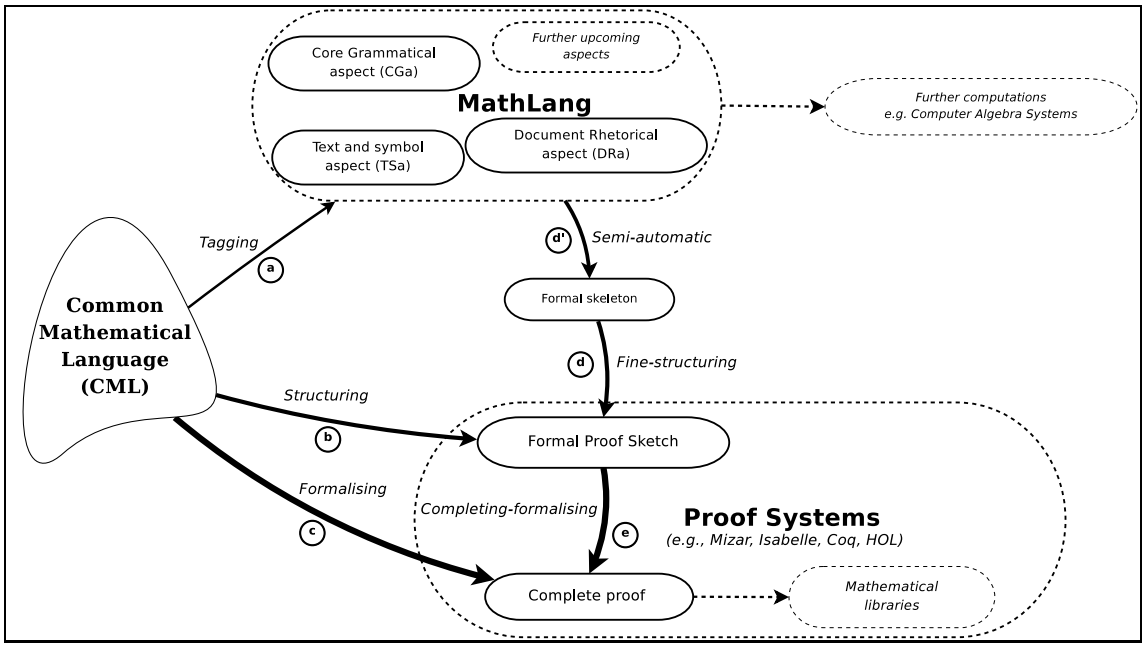


Figure 2.5: The MathLang approach to computerisation/formalisation.
The figure presents an overall situation of work in MathLang. At the time of writing this thesis, the MathLang framework provides three developed aspects.
The labeled arrows shows the computerising paths from CML to any *proof system*. In the current state of MathLang we are trying to provide a full formalisation path from CML via MathLang to Mizar (by K. Retel), Isar (by R. Lamar) and Coq (by C. Zengler). Part of the path from MathLang to those *proof systems* still has a rough shape and is not fully developed. In this thesis, we mainly focus on the path ⓐ-ⓓ-ⓔ. We also briefly compare it with the path ⓑ-ⓔ and the path ⓒ. The path to Mizar has been done through a number of encodings and has a number of formal rules.
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.

malised and developed three aspects: CGa (Core Grammatical aspect), TSa (Text & Symbol aspect) and DRa (Document Rhetorical aspect). Each of those aspects has focus on a different knowledge layer of a common mathematical document.

2.3 Core Grammatical aspect

The Core Grammatical aspect (CGa) is a formal language initially inspired by MV and WTT and derived mainly from WTT. The CGa has taken the best features of MV and WTT, simplified the difficulties of WTT and enhanced a number of WTT constructions. As mentioned above, the CGa continuously evolves and shapes by experimenting on further mathematical texts. This section presents the current version of CGa and the whole MathLang framework, at the time of writing the thesis.

2.3.1 The CGa Grammar and Language Description

In the following subsections we present the CGa constructs on the language level. We also introduce an its abstract syntax.

2.3.1.1 Notation convention

We use a `typewriter` font to present MathLang examples.

For identifiers without parameters we omit the parentheses. For instance for `x` we write `x` in place to `x()`, and `x:set` in place of `x():set`.

We abbreviate category expressions to shorten the syntax of some term, set and noun categories. For example, `term`, `set` and `noun` are abbreviations for `term(Noun{})`, `set(Noun{})` and `noun(Noun{})` respectively, which all use the *noun description* (`Noun{`s`}`)

We also omit a double parenthesis when introducing new identifiers or definitions by the use of *adj descriptor* (i.e., `Adj(`e`){`s`}`) and *noun descriptor* (i.e., `Noun{`s`}`). For instance, we write `Adj(e){x:term}` instead of `Adj(e){{x:term}}`, and `Noun{x:term}` instead of `Noun{{x:term}}` respectively.

2.3.1.2 *Step*

The basic construct of CGa are *step* and *expression* (which asserts a truth). The WTT constructions like book, line, declaration, definition and statement are all captured and represented as a *step* in CGa. For example: `irrational(sqrt(2));`, `n:N;`, `Th:=implies(subset(A,B),subset(intersection(A,B),B));`.

2.3.1.3 *Block*

The new construction in CGa versus WTT is a *block*. A CGa *block* is simply a collection/sequence of *steps*, i.e., $\{s_1, \ldots, s_n\}$. For example a sequence of reasoning steps could be encapsulated and annotated as a *block* in CGa. Here is for example a *block*:

```
{
  (x + y) + 1 =  (x + y)';
  (x + y)' = x + y';
  x + y' = x + (y + 1);
}
```

2.3.1.4 *Local scoping*

A step can be a *local scoping*, annotated as: $s_1 \rhd s_2$. A *local scoping* puts a *step* (i.e, s_1) as a context for the development *step* (i.e., s_2). The scope of definitions and declarations of *step* s_1 are restricted in range to *step* s_2. For example `a:R |> =(+(a,b),+(b,a));`, the scope of a variable `a` is limited to the following *step*, i.e. `=(+(a,b),+(b,a));`, and is not available after this *step*.

2.3.1.5 *Definition*

As mentioned above, a CGa *step* can also be a *definition*. A *definition* assigns to a particular expression a shorthand name. Moreover, a *definition* could be parameterised, for example: `a:R |> Id(a) := a;`.
Another example, the subtraction of two sets $A - B = \{x \mid x \in A \wedge x \notin B\}$, could be defined with two parameters A and B:

```
{A:set; B:set;} |>
   subtraction(A,B) := Set(x:term, and(in(x,A),notin(x,B)));
```

CGa refined the MV and WTT concept of *definition by cases*. A *definition by cases* make explicit all the *cases* of the definition. For example,

```
{ x:R; y:R;
    <(x,y) |> max(x,y) := y;
    =(x,y) |> max(x,y) := y;
    >(x,y) |> max(x,y) := x;
}
```

2.3.1.6 *Declaration*

A CGa *declaration* introduce a new identifier by providing its arguments and result, and assigning to parameters and result their categories. For example, `x:term`, declares an identifier `x` and assigns its category to be `term`. Similarly, examples like `G:group` or `ABCD:rectangular` introduce new objects, where `group` and `rectangular` represent nouns. These examples present two ways of expressing the belonging of an identifier to a grammatical category. This is used by either providing explicitly the grammatical category (e.g., `N:set`), or by providing either a set (e.g., `x:N`, where `N` is a set of natural numbers) or a noun-expression (e.g., `G:group`).

Furthermore, the example `subset(set,set):stat;` is a declaration of the operator `subset` (represented as $\subset$ in CML) which takes two arguments of type **set** and yields a result of type **stat**, i.e., a statement.
A CGa *declarations* can also be used when declaring *binders* like the $\forall$ and the $\exists$ quantifiers or the *Set* binder, see table 2.1.

Declarations	
Common name	CGa declaration
Set binder	`Set( dec(term), stat ) : set`
$\forall$ quantifier	`forall( dec(‘a), stat ) : stat`
$\exists$ quantifier	`exists( dec(‘a), stat ) : stat`
Instances	
Formula	CGa equivalent
$\{x \in \mathbb{R} \mid x^2 = 4\}$	`Set( x:R, =(power(x,2),4) )`
$\forall P, P(S) \Rightarrow P(\mathbb{N})$	`forall( P(set):stat , =>(P(S),P(N)) )`
$\exists n \in \mathbb{N}$ st. $n < 2$	`exists( n:N, <(n,2) )`

Table 2.1: Examples of binder identifiers

2.3.1.7 *Noun description*

CGa introduces two new approaches for introducing *nouns* and *adjectives*, i.e., a *noun description* and an *adjective description* respectively.

A CGa *description* is an expression of the form `Noun {`s`}`, where s is *step*, that could be a single *step* or a *block*. Such *description* is called a *Noun descriptor*, which is a collection or a class of all entities with specific *characteristics* defined by a step or a sequence of steps. For example, `Noun {center:term}` is *Noun descriptor* which provides a characterisation for all entities that have a `center`. Translating it further, the example is a set of all objects that possess a `center`, e.g., figure, triangle, rectangular, segment, etc. If we want to be more specific and make the range smaller we need to introduce additional restrictions within the *step* of the *Noun descriptor* or to apply a CGa adjective to the introduced Noun.
For example, the listing below introduces a `segment` using a CGa *noun descriptor*.

```
segment := Noun {
                    length: term;
                    is_finite(self.length);
                    =(card(ends),2);
                  }
```

A *Noun descriptor* introduces a *noun*. Moreover, a *noun* has a *character* described within the *step* of the *Noun descriptor*. The *character* can be seen as a direct counterpart of *field* or *method* in the object-oriented language jargon. The CGa development makes this direct correlation of CGa *Noun* to a *class* in object-oriented jargon, due to the limitation of MV and WTT annotations.
Let us consider that `circle` is a CGa *Noun* introduced using the *Noun descriptor*, and possesses a *character* called `diameter` representing the diameter of a circle. Then for any term `C` being `circle` (i.e., `C:circle;`), a `C.diameter` stands for the diameter of the circle `C`, annotated in CML as ⌀`C`.

It is important to remark that there exists a category constructor *noun*, which is used for building category expressions. In the following example, three identifiers with one character `x` are defined: `a` is a noun, `b` is a term instance of a noun and `c` is defined as a noun using *Noun descriptor*.

```
{
  a : noun( Noun{x:term});
  b : Noun{x:term};
  c := Noun{x:term};
}
```

2.3.1.8 *Adjective description*

CGa introduces another *descriptor* called *Adjective descriptor*. It is an expression of the form `Adj(`*e*`){`*s*`}`, where *e* represents any *expression* and *s* annotates a CGa *step* (similar to a *Noun descriptor*). An *Adjective descriptor* defines an *adjective*, which shrinks or extends a noun expression, that is the expression `exp` placed in parenthesis of the *adjective descriptor*, to form a new noun expression and makes the characteristic of the noun more specific (when shrinking) or wider (when extending the noun expression). The *Adj* constructor takes as a parameter the noun to be extended to form a new noun. For example,

```
rectangle := Noun{ =(card(self.sides),4);
                   forall(A:angle, =(degrees(A),90));
                 };

equilateral :=
   Adj(rectangle) {
                    forall(sideX:sides,
```

```
                        forall(sideY:sides,
                          =(sideX.length, sideY.length)));

                    };
square := equilateral rectangle;
```

The above listing defines a `rectangle` as a *noun*, and introduces an *adjective* `equilateral`. The last sentence of the listing forms a new *noun*, as a composition of *adjective* and *noun*. We can see that *adjectives* play the role of functions from *noun* to *noun* (i.e., *adjective* : *noun* $\rightarrow$ *noun*). In the CGa system where *nouns* are classes, the *adjectives* are mixins [FKF98] in the object-oriented jargon. A mixin is a function from class to class. As in mixin calculus, a mixin forms a new class, hence a mixin could be applied to a mixin as well. Therefore, in CGa an adjective can be applied: to an adjective to form a new adjective, or to a term to form a new term, or to a set to form new set. In CGa we call these constructions *refinements*. The following example represents refinements of two adjectives, that are defined and refine the noun-expression `group`. The adjective `finite` states that the set E of group is finite, whereas the adjective `Abelian` states that the operator of the group is commutative.

```
Group := Noun{
            E:set;  op(term,term):term;

            forall(a:E, forall(b:E, in( op(a,b), E);));

            forall(a:E, forall(b:E, forall(c:E,
                =( op(op(a,b),c) , op(a, op(b,c)) )
            )));

            exists(e:E, forall(a:E,
                      =(=( op(a,e), op(e,a)), a) ));

            forall(a:E, exists(b:E,
                      =( =(op(a,b), op(b,a)), e); ));
             }

finite := Adj(Group) { finite_set(E); };

Abelian := Adj(Group) {
```

```
        forall(a:E, forall(b:E,
            =(op(a,b), op(b,a));  ));
    }
```

We can combine these two adjectives to obtain a new expression `finite Abelian Group`. The combination of these adjectives could also be used to form another expression of `Abelian finite Group`. However both expressions are the same in CGa and share the same type of **noun**.

It is important to remark that there exist a category constructor *adj* (i.e., `adj(exp1,exp2)`, which is used for building category expressions. In the following example, two identifiers are defined: `a` is an adjective (declared as a category expression), and `b` is defined as an adjective using the *Adj descriptor*.

```
{
  a : adj(Noun{x:term}, Noun{y:term});
  b := Adj(Noun{x:term}){y:term};
}
```

The following example `right_angled: adj(square, rectangular)` represents a declaration of an adjective using the *adj* category expression.

2.3.1.9 *Category expression*

CGa uses grammatical *categories* to make explicit the grammatical role played by the elements of a mathematical text. CGa has identified and allows to use *category expressions* like: *term, set, noun, adj. stat, dec.* In many cases of the above examples we were using the *categories* to introduce an instance of the required category.

2.3.2 CGa Types and Type System

The first aspect of MathLang framework consists of:

1. clear and concise grammar and language
2. types and associated type system

The grammar and informal language of CGa has been described in Section 2.3.1. This section represents the CGa types and the checking system.

term	Common mathematical objects.	"x^2, $\angle ACD$, "
set	Sets of mathematical objects.	"$\mathbb{R}$, $\emptyset$, $A = \{1, 3, 4, 7, ...\}$"
noun	Families of **term**s.	"group, polygon"
adjective	**Noun** refiners.	"infinite, positive, parallel"
statement	Affirmations, arguments, properties, assertions, ...	"$1 \geq -2$, $a + 0 = a$"
declaration	Introductions of new symbols or notions.	"Let a be ..."
definition	Explanations of the meaning of new symbols notions.	"A positive number is ..."
step	A group of mathematical assertions.	"..., therefore ..."
context	Preliminary assertions prior to a **step**.	"Assume ..."

Table 2.2: TSa box annotations' colour coding system

2.3.2.1 Grammatical categories/types

As mentioned in Section 2.3.1.9, CGa uses a finite set of grammatical *categories* to capture the structure and common concepts used in CML. The aims are:

- to make explicit the grammatical role played by the elements of a CML texts,
- to allow the automatic validation of the grammatical and the reasoning structure of a CGa encoding corresponding to a CML text.

The finite set of grammatical *categories*, together with a short explanation and examples, is presented in table 2.2. The first column of that table, represents the colour coding for CGa categories. This colour coding, also presented on Figure 2.6, will be used across the thesis to represent examples, encoding of CML documents and to explain further aspects of mathematical documents. The same colour coding is used within the implemented user friendly text editor which allows annotating CML texts within the MathLang framework.
The second column of table 2.2 provides a short explanation of each *category*, whereas the third column presents few CML examples of each *category*.

The types of CGa are more complex and sophisticated than the *weak types* of WTT, although they still have number of limitations.

To give a flavour of the types and the typing system, let us consider the CML example: $x \cdot 1 = x$. The CGa encoding of the example could be translated into the following format:

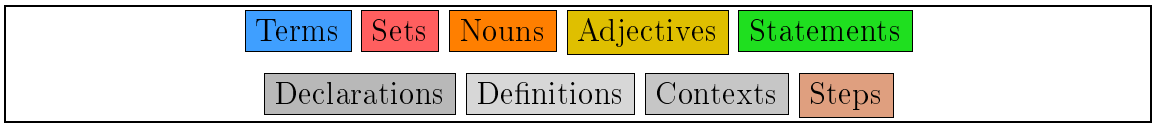


Figure 2.6: MathLang's colour coding system for CGa's grammatical categories.

```
{
  x: term;
  1: term;
  multiply(term, term): term;
  equal(term, term): stat;

  equal( multiply(x,1), x );
}
```

So the actual expression $x \cdot 1 = x$ is encoded in CGa as `equal( multiply(x,1), x );`. Before we encode the CML example in the CGa grammar we are obliged to declare or define identifiers in the context (or so-called preamble) of the encoding. If we skip the declaration steps of each identifier and operation, or omit the declaration of one of identifiers, then the CGa checker will prompt an error saying that the identifier is unknown.

The example of `equal( multiply(x,1), x );` is written in a language with a type inference. The type of `equal, multiply, = , x` and `1` would be inferred or retrieved, from the earlier declaration of identifiers, by a type inference system. The explicit typing of the statement example is represented as follows:

```
( equal: term -> term -> statement )
    ( (multiply: term -> term -> term) (x:term) (1:term) )
      (x: term)
```

The CGa checking system captures errors like an identifier being used without a prior proper introduction, or the wrong number of arguments being given to a function, or a wrong type of identifier being used within a function declared earlier. The CGa typing system derives typing judgements to check whether the reasoning parts of a document are coherently built. It is important to understand that the goal of the CGa's typing system is *not* to ensure the full logical correctness of an encoding, but merely to check the well formedness of the encoding according to the typing rules provided by the CGa.

The typing system as well as the types were developed, revised and built by F. Kamareddine, M. Maarek and J.B. Wells, during M. Maarek PhD course. The implementation of the typing system is primarily by M. Maarek. The exhaustive description of the typing system and the typing rules is presented in [Maa07]. The thesis also presents the typing system with a huge cover of small examples of CML texts encodings.

2.4 Text & Symbol aspect

This section briefly describes another knowledge level of a mathematical document that is captured by encoding the document within the MathLang framework. We call this **aspect** a Text & Symbol aspect (TSa).

The Text & Symbol aspect (TSa) builds the bridge between a CML text and its grammatical interpretation. The TSa adjoins to each CGa expression a string of words and/or symbols which aim to act as its CML representation.

The CGa grammar provides computable constructions to represent mathematical reasoning. The TSa annotation adds to this strict language information additional levels of information, mainly how each CGa element should be printed on paper or on screen. This makes MathLang's encoding of mathematical texts faithful to the traditional mathematical authoring [KMW04a]. TSa adds on top of a CML text a new dimension to the document. This dimension is rendered with a finite number of coloured boxes (annotating the CML symbols) following the colour coding system of Figure 2.6. The TSa allows rendering pieces of CGa encoding with pieces of CML in the form of mixture of natural language, symbolic formulas and formal CGa interpretation. We present the TSa usage via a number of examples throughout the thesis.

2.4.1 Annotation example

The colour boxes of each example are added by the MathLang user himself. The implementation of the TSa **aspect** is done in *plugin* for TEXMACS, a brief explanation of that is provided in section 2.4.2.

The MathLang annotation process follows the following simple rule:

> to each semantic entity of a document the author has to attach one of the CGa grammatical types and an identifier (a string) to be used as the CGa interpretation.

The string value could be arbitrary, whereas the grammatical *category* has to be the best matching to the CML symbol/expression. Although the *category* choice and assignment depend on the author's understanding of the text.
For example, <X> X, represents an annotation of "X" being a set, using the TSa annotation system. The background colour informs the grammatical *category*, which in our example is the *category* **set**. The CGa interpretation attribute is a value introduced as superscript, on the left hand side of the annotation box, between the < and > angle brackets i.e., <X>. In our example the CGa interpretation symbol is exactly the same as the CML symbol.

As a small example of the annotating process of the TSa aspect, consider the following CML sentence:

Exists an element x in $\mathbb{R}$ such that $x \geq 2$.

Figure 2.7: A CML example text used to present the TSa annotation process.

Considering Table 2.2, we annotate x as **term**, and provide an x as a CGa interpretation when annotating the example 2.7. So the annotation looks like:

Exists an element <x> x in $\mathbb{R}$ such that <x> $x \geq 2$.

Next we annotate the number "2" as **term** and R as a *category* **set**, providing the same symbol names for the CGa interpretations.

Exists an element <x> x in <R> $\mathbb{R}$ such that <x> $x \geq$ <2> 2.

Next note that symbol "$\geq$" is a predicate which takes two arguments of type **term** and returns **statement**, i.e., $term \times term \rightarrow statement$.

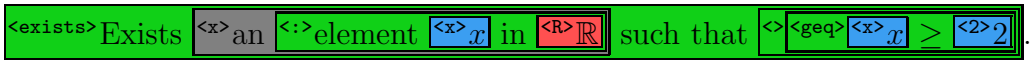
Exists an element <x> x in <R> $\mathbb{R}$ such that <geq> <x> $x \geq$ <2> 2.

Next we annotate the element-hood relation, which is expressed in CML as "an element x in $\mathbb{R}$". This predicate can be presented in CML as well known symbol: $\in$, which takes two arguments the first of type **term** and the second of type **set**, in our example, and returns a **statement**.

Exists an <:> element <x> x in <R> $\mathbb{R}$ such that <geq> <x> $x \geq$ <2> 2.

Next we notice that in the example we use the quantifier (or the binder) "exists" (in CML it is presented as the symbol $\exists$). The binder could be presented in the form

of: "∃ [part1]. [part2]", whereas "part1" is a **declaration** for "part2" which is a **statement** using identifiers declared in "part1". The "exists" quantifier annotates a new **statement**. In our example we annotate this quantifier as follows:



Finally, if the statement is a step in a reasoning block, we annotate this as a **step** in CGa. The CGa interpretation annotation could be omitted or can be provided with a short symbol name `s1`.

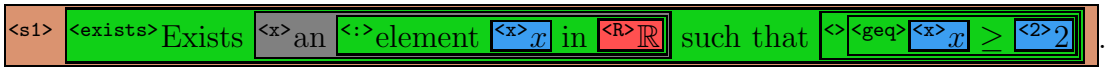


Figure 2.8: The TSa annotation of the CML example presented on Figure 2.7 with displayed CGa interpretation identifiers.

It is important to notice that most of the CGa interpretation symbols and predicates used in the sentence, like `x`, `2`, `R`, `:`, `geq`, `exists`, are introduced somewhere at the beginning of the document with either a **declaration** or a **definition** (see table 2.2). Further description of the details and requirements for annotations and CGa interpretations symbols could be found in [Maa07].

Hiding the CGa interpretation from Figure 2.8, provides the presentation of the CML example with the TSa annotation only together with a colour coded boxes, see Figure 2.9

Figure 2.9: The TSa annotation of the CML example (Figure 2.7) with hidden CGa interpretation identifiers.

Similarly, from the TSa encoding presented on Figure 2.8 we can extract the following CGa portion:

```
exists(x:R, >=(x,2));
```

2.4.2 Implementation in a nutshell

The idea of the TSa representation is independent of the visual formatting language. In the above examples we use TEXMACS as an authoring tool. The coloured boxes

shown in Figure 2.9 are added by the MathLang user himself. The implementation of TSa is done as a *plugin* for the scientific text editor TEXMACS [12]. The view of our example shown in Figure 2.9 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. After verification, the CGa checker returns errors captured during verification (e.g., missing argument, wrong category assignment etc.). These errors are later on rendered in the original TEXMACS document using the user friendly presentation of the same fashion surrounded boxes with appropriate error number attached to each part of the sentence which cause the error. The description of each error is stored in the MathLang menu within the TEXMACS editor. An example and comprehensive explanation of the MathLang CGa+TSa checker is written in M. Maarek PhD Thesis [Maa07].

The approach of such implementation has lots of advantages. The main advantage is that the author does not need any extra knowledge to be able to annotate a document using the MathLang framework. The author has to use mouse and keyboard commands to annotate the CML text entered in TEXMACS with boxes representing the CGa grammatical categories 2.2 following the colour coding system of Figure 2.6. Additionally, the author has to assign CGa identifiers as an interpretation. Both, the colour coded boxes together with the CGa assigned identifiers explicitly indicate mathematical meanings.
Furthermore, the author is not forced to understand any internal representation of the TEXMACS document with annotated TSa dimension on top of it. Moreover, the internal MathLang representation of the document is not presented to the user or the author, although it is available for viewing at any time during the annotation process. The formal XML presentation of a MathLang document is awkward to read and understand without a distinctive knowledge of the XML language and the MathLang XML schema.

The transformation process of TSa annotated document is done "invisibly" to the author. Similarly, the CGa verification is done beyond the editor, and results (any errors captured, mistakes etc.) are rendered on top of the TSa annotation and are presented in the TEXMACS document. The visualization of error messages is straightforward and does not require any additional knowledge to fix them by the author.

Summarizing, the MathLang framework does not require any technical knowl-

[12] `http://www.texmacs.org/`

edge from the author willing to annotate his document within MathLang. Moreover, the internal presentation of the MathLang annotation is not intended to be read by the author.
Furthermore, the MathLang user interface allows displaying different presentation views to the author. It allows displaying either a pure CML view which hides coloured boxes of TSa and CGa identifiers annotation, a pure CGa view, or various combined views with coloured boxes, grey scheme boxes, without boxes, CGa information etc. The same interface allows adding "*souring*" rules that are described in Section 2.4.3.

All that, makes the MathLang framework unique in comparison with other existing tools designed for the computerisation, formalisation and verification of common mathematical documents.

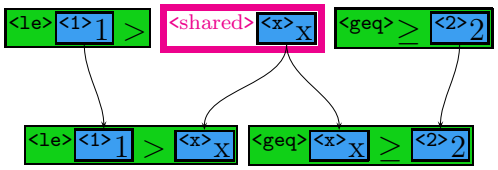
2.4.3 The *Souring* facility of the TSa

There exist a number of styles of writing mathematical texts which are mainly subject to the author style of writing mathematics. Therefore, there are certain challenges in CML constructions and formations that have to be handled using the TEXMACS *plugin* for MathLang.

The TSa development has been carried out to the level which gives the users the ability to write naturally and concisely, using different formations and constructions. Such development resulted in a number of annotation rules and transformation rules, that could be found in [KLMW07, Maa07].

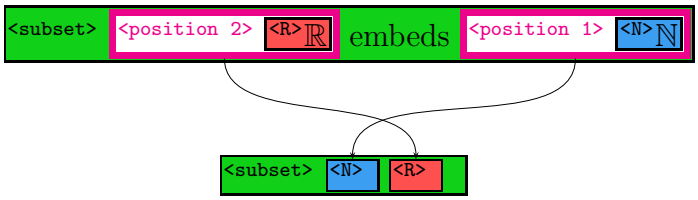
For example, in the text "$1 > x \geq 2$", the term "x" is shared between two equations: "$1 > x$" and "$x \geq 2$". This is problematic while annotating using the TEXMACS *plugin*. While annotating such an expression one would like to simultaneously create the annotation of "<le><1>1</1>><x>x</x></le>" and "<geq><x>x</x>≥<2>2</2></geq>", whereas the term "<x>x</x>" is shared among both expressions. Unfortunately, these two statements cannot overlap. Therefore, the additional feature called "*souring*" has been developed to facilitate such constructions. The term *souring* is an opposite to the commonly used, especially in computer language jargon, *sugaring* (e.g., syntax sugaring).

In the above example, the solution to make use of the term "<x>x</x>" within the two expressions simultaneously, is to use the *share* rewriting method from the *souring* facilities. With this approach, the expression would be annotated as follows in the first line, and would be automatically rewritten as indicated in the line below.

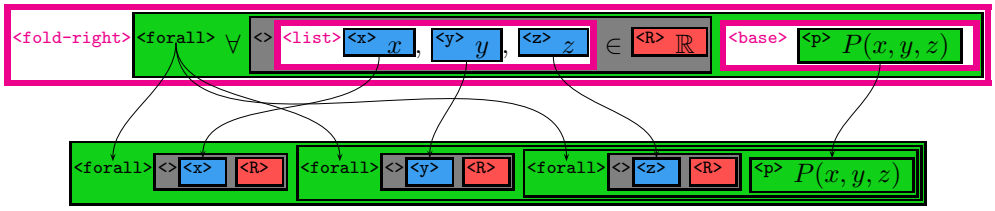


As we can notice on the example, the *souring* annotation is presented as a third kind of node label used in TSa, in addition to the colour boxes and superscript stating the CGa identifiers interpretations.

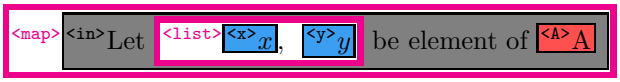
The expression "$\mathbb{R}$ embeds $\mathbb{N}$" or "$\mathbb{N} \subset \mathbb{R}$" should both be interpreted as `subset(N,R)`. For indication of the order of arguments for the first expression within the TSa colour boxes annotation we use a *position souring* rewriting rule. The annotation conforms of "$\mathbb{R}$" and "$\mathbb{N}$" with `position 2` and `position 1`, respectively. The example below provides a visual presentation of the example with a *position* rewriting rule.



Further *souring* rules have been developed to support more sophisticated CML formulations. For example support for folding and mapping over lists has been developed. For instance, folding over the list of declared variables under the "forall" quantifier of the form $\forall x, y, z \in \mathbb{R}.P(x, y, z)$ is a shorthand for the expression: $\forall x.\forall y.\forall z.P(x, y, z)$.



Similarly, for instance, the mapping of arguments of a declaration statement "Let x, y be element of set A" is a shorthand for the statement: "Let x be element of set A and let y be element of set A".



The design and development of the Text & Symbol aspect of the MathLang framework is due to F. Kamareddine, M. Maarek and J.B. Wells with contribution by R. Lamar to the souring rules [KLMW07, Maa07]. The current, on the time of writing the thesis, implementation of the MathLang is primarily by M. Maarek.

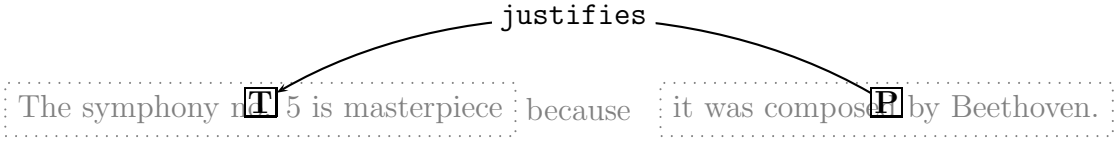
2.5 Document Rhetorical aspect

Apart from the CGa and TSa aspects of mathematical documents, we have identified an additional aspect called Document Rhetorical aspect (DRa). The DRa extends the knowledge already computerised in CGa. It focuses, as the name suggests, on the rhetorical aspect of mathematical documents. It is used to describe and annotate chunks of texts according to their narrative role played within the document. The DRa makes explicit 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** or **block** construction. One would encode *chapter*, *section* or *lemma*, *theorem* by this unique **step** construction.

To enhance flexibility, CGa does not differentiate between narrative 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. 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* of chunks of text within the document. 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, relations between recognised chunks of text sometimes stay implicit in the original document. DRa annotations allow to express such information explicitly, since the DRa works as an annotation system for the CGa **step**.

To illustrate this aspect let consider the statement, "The symphony no. 5 is masterpiece, because it was composed by Beethoven.". The first part of the statement "The symphony no. 5 is masterpiece" is an assertion, which can be annotated and labelled as theorem. The author has the freedom to attach for this

selected chunk of text any preferred label. The word "because" is an indication that the second part of the statement, "it was composed by Beethoven", is a kind of justification. If we call the former a theorem marked `T` and the later proof `P`, we can relate them using the RDF[13] triples notation as `[P, justifies, T]`. The diagram below presents the relationship between such annotated assertions.



The DRa annotation uses a finite set of binary predicates to annotate relationships among annotated chunks of texts. This includes an above relation *justifies*. It is important to note that the DRa does not require the logic of each statement to be sound. It only requires that the argumentation is acyclic and that some justification is offered if necessary for each claim. Further description of the DRa system will be discussed in Chapter 4.

The Document Rhetorical aspect of the MathLang framework is the main contribution of this thesis. Its development, comprehensive description and discussion is presented in Chapter 4.

2.6 Conclusion

In this chapter we introduced the *common mathematical language* term which stands for a mathematical language that we are taught from the early years of secondary school to the university level. We also presented existing tools for computerising and formalising mathematics. We discussed different approaches of existing *proof systems*. Finally we gave an overview of the MathLang project and its aspect oriented design. Furthermore, we presented MathLang origins, goals and current overall situation of development. We also illustrated the MathLang framework in terms on encodings and covered that illustration with a number of examples annotated in MathLang.
In Section 2.5 we shortly described the Document Rhetorical aspect which is a part of MathLang aspect-oriented design. The DRa aspect is the main contribution of this PhD and has been developed during this PhD studies.

[13]Resource Description Framework, described in [Con04, LS99]

Chapter 3

The Mizar Project and Some of Our Formalisations in Mizar

In this chapter we present a short description of the Mizar project. Section 3.1 presents a brief overview of the whole project, where we discuss the overall goals of the project, different tools and third-party development approaches. In Section 3.2 we discuss the Mizar Language and its similarities to the CML. The following Section 3.3 describes the Mizar *Article* which stores the authors input of the mathematical knowledge in the Mizar language and which the Mizar system verifies. In Section 3.5 we present the Mizar Mathematical Library (MML), which is the biggest collection of computer verified mathematical facts. Primarily, the MML is a library of Mizar articles which form a graph of connected and related mathematical facts that form the integral library. Moreover, the library is centrally maintained and upgraded very often.

In the same section we provide a general overview of a number of complex theories and books formalisation in Mizar. Furthermore, we present the MML Query a semantic search engine, that was built to provide semantical search over the Mizar Mathematical Library. In the same section we shortly present the Mizar types structure.

The following Section 3.6 discusses formalisation attempts done by K. Retel. Those formalisations resulted in a number of articles submitted in the MML and later on automatically translated into English language and published in the Formalized Mathematics journal.

Section 3.7 presents a short description of history use of the Mizar system in a number of teaching experiments conducted at many universities. K. Retel has been involved in some of those experiments and he placed important role of tutor

and teacher of the Mizar system.

Finally in Section 3.8 we present the Formal Proof Sketch (FPS) term, invented by F. Wiedijk, which applies to any formal proof system and language, and particularly to the Mizar.

It is important to note that this chapter is mainly a survey about the Mizar. This means that some parts of the content of this chapter might contain a direct or indirect relations, citations, examples and express similar or very close ideas, thoughts and explanations that were written in a number of publications regarding the Mizar. The chapter also contains number of short examples taken directly or indirectly from the MML.

During the course of the PhD of this student, we had used many versions of the Mizar system. This was due to a rapid enhancements and development of the software, as well as lots of submissions, revisions and major improvements of the Mizar library (Mizar Mathematical Library). Each time we present bigger examples of formalised texts in Mizar language it should come from the latest, at the date of writing the thesis, version 7.9.03 of the Mizar system, which is distributed with version 4.108.1028 of the Mizar Mathematical Library. If the example was verified and used different MML version, we mention the version of the Mizar system and the MML on which this example was composed. This is presented usually as a footnote to the main example. We will talk about the Mizar by providing number of toy examples from original articles taken randomly from the MML. We would also use some articles that have been written by this student, K. Retel, and have been published in the MML and electronic journal FM.

3.1 Overview of The Project

The main goal of the Mizar project[1] is to create a system for computer-aided formalisation of mathematics [Try77, Try78, Try80, Try82, RT99, Rud92, MR05]. The project started in 1973 at the University of Warsaw, Poland, when A. Trybulec envisioned *computer assistance* in the process of writing mathematical papers. The idea was motivated in the latest stages of writing A. Trybulec's PhD [MR05]. The name, Mizar, of the project appeared in late 1972, picked up by the wife of A. Trybulec. A. Trybulec asked her for a good name of a project, while she was looking through an astronomical atlas, and she suggested Mizar, the name of a star in the familiar Big Bear constellation [MR05].

[1] `http://mizar.org`

Since 1973, A. Trybulec has been leading the group of mathematicians that forms the Mizar Group. The Mizar Group mainly refers to people from the University of Bialystok, which is the heart of the Mizar project. The ongoing development of Mizar has resulted in several things:

1. the Mizar language,
2. the Mizar system,
3. the Mizar library,
4. Mizar software utilities for working with Mizar documents and the Mizar library,
5. and an electronic hyper-linked journal of mathematics that is also published in the paper edition.

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 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 [Jaś34]. 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 development of the Mizar language resulted in the Mizar system. The system allows to check mathematical publications written in the Mizar language and to verify the logical structure and reasoning of such formalised documents.

The essential achievement of the project is a centrally maintained library of mathematical knowledge (the Mizar Mathematical Library – MML). This corpus of formalised mathematics is accompanied to the Mizar system and it is distributed with the system. Moreover, the MML library together with the Mizar system play the integral unit. For many years to nowadays, the Mizar Mathematical Library is the biggest collection of digitalized mathematical texts verified by computer [Wie03].

The MML consists of Mizar documents, which are called *Articles* within the Mizar community. This library is based on two axiomatic *Articles*: `HIDDEN` [Com89] which consists of built-in notions, and `TARSKI` [Try89] which presents axioms of

the Tarski-Grothendieck set theory [Tar39]. All other *Articles* of the 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. At the time of writing this thesis, the library includes 1011 *Articles* contributed by 209 authors[2], a number of whom have been active on a long term basis. There is a number of introductory papers and manuals on Mizar [Miz, RT99] as well as practical hints for writing Mizar *Articles*.

3.2 The Mizar Language

The development of Mizar started with the proposition of a formal language for editing mathematical papers. This Mizar language, on its early days (November 1973), was intended to be a formal language for recoding mathematics in such a way that [MR05]:

- the papers could be stored in a computer and later, at least partially, translated into natural languages,
- the papers would be formal and concise,
- it would form a basis for the construction of an automated information system for mathematics,
- it would facilitate detection of errors, verification of references, elimination of repeated theorems, etc.,
- it would open a way to machine assisted education of the art of proving theorems,
- it would enable automated generation of input into typesetting systems.

Summarising, the main goal for the original Mizar language design was to be close to the mathematical vernacular used in publications with the requirements

[2]statistics are taken from the web: `http://merak.pb.bialystok.pl/` – last accessed on 2008-08-15, MML version: 4.100.1011.

that the language has to be simple enough to enable computerised processing, in particular mechanical verification of correctness [BR02].

Therefore, the goals and the main stress placed on editorial work resulted in the development of the Mizar language to its current state where it mimics the Common Mathematical Language and provides a way that mathematicians work. The language is based on first-order logic and proofs are written in a style of natural deduction as proposed by Jaśkowski [Jaś34]. Moreover, the Mizar language is the most accessible of the *formal languages*, in terms of readability and write-ability, by mathematicians, as expressed by the author A. 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. *[RT99, pp.2]*

In summary, the Mizar language is a tiny subset of English words frequently used in the CML. It consists of 102 reserved words like: `define, let, be, if, where, assume`, which form a lexicon of basic tokens of the language. Further tokens are defined by the author in *vocabularies* and are shared among a number of Mizar articles in the Mizar Mathematical Library (MML). Similarly, common words in mathematical texts, like `assume, consider` and `exists` are part of the Mizar language. Words that link sentences like `then, thus` and `hence` are part of the Mizar language as well. All these reserved words are used in an intuitive way when writing Mizar article.

As an example of the Mizar language let us look at the following CML theorem from Set Theory.

Theorem 1. *For any sets X and Y, it holds that $X \cup Y = \emptyset \Longrightarrow X = \emptyset$.*

The above sentence can be translated into the Mizar language and represented in the following manner:

```
reserve x,A,B,X,X',Y,Y',Z,V for set;

theorem :: XBOOLE_1:15 :: BOOLE'59:
  X \/ Y = {} implies X = {};
```

Listing 3.1: An example of Mizar theorem statement.

A more complex example of the CML fragment shown in Figure 3.1 is taken from the work [Tho00] by S. Thomasse. This fragment introduces the definition of binary relations *embeding*.

A binary relation $R = (V, E)$ is *embedded* into another binary relation $D = (W, F)$ (or S *embeds* R) if there is an injective mapping f from V into W such that $(x, y) \in E$ if and only if $(f(x), f(y)) \in F$.

Figure 3.1: A more complex example of the definition written in CML and taken from [Tho00].

The definition from Figure 3.1 could be represented in the Mizar language as follows (Listing 3.2):

```
definition
  let R,S be RelStr;
  pred S embeds R means
:: NECKLACE:def 2
  ex f being Function of R,S st f is one-to-one
  & for x,y being Element of R holds
  [x,y] in the InternalRel of R iff
                                [f.x,f.y] in the InternalRel of S;
end;
```

Listing 3.2: A translation of the original text from Figure 3.1 in the Mizar Language. This is taken from the Mizar article written by K. Retel and published in the MML [Ret03a]

The definition presented in Listing 3.2 is part of the Mizar article written by K. Retel, which is the first article of the series of articles [Ret03a, Ret03b, Ret04] attempting to formalise the original paper [Tho00]. Further description of this attempt and other examples of the formalisations done by K. Retel are discussed in Section 3.6.

As we can see from the above Mizar text fragments, the Mizar language mimics the common mathematical language and more importantly it matches all the goals of the formal language Mizar (stated at the beginning of this section). Moreover, the Mizar Language could be read and understand by an average mathematician without requiring an exhaustive knowledge of Mizar. One of the visible differences between the Mizar and the CML representations of the same fragment are: (1) the usage of different notions for a number of identifiers and operations, (2) the rigid structure of formalised text. Some of those identifiers are reserved words, like `definition, let, be, means, pred, implies, holds`, where others are intro-

duced and defined in other Mizar articles, like `Function of R,S, one-to-one, Internal Rel of R, [, f., {}`.

Table 3.1 presents the notation of some common symbols found in mathematical documents, and expresses them against their Mizar counterparts. This provides quite generic view over the Mizar language symbols used to construct statements.

Mizar:	**CML:**	**Mizar:**	**CML:**
`&`	$\wedge$	`in`	$\in$
`or`	$\vee$	`{}`	$\emptyset$
`implies`	$\Longrightarrow$	`c=`	$\subseteq$
`iff`	$\Longleftrightarrow$	`=`	$=$
`not`	$\neg$	`c<<`	$\subset$
`for a being A st P[a] holds R[a]`	$\forall a : A(P.a \Longrightarrow P.b))$	`\/`	$\cup$
`ex a being A st P[a]`	$\exists a : A(P.a)$	`/\`	$\cap$
		`\`	$\setminus$

Table 3.1: Comparison of some common mathematical symbols and their presentation layout in the Mizar language

.

In the Mizar language we have some blocks of text that are named in the same way as compared to the CML counterparts. For instance, we have a `Definitional-Block`, as expressed in Listing 3.2, or a `theorem` as shown in Listing 3.1. Similarly to any proof system or programming language, the Mizar language has also comments. Comments in Mizar are expressed as texts displayed after the symbol "`::`". For instance, in Listing 3.1 string: `XBOOLE_1:15 :: BOOLE'95` is treated as a comment, although it also expresses the article name from which the theorem is taken, i.e., `XBOOLE_1` and the number of the theorem from that file, i.e., `15`. We can refer to that theorem in other articles by using this name in the justification reference list, for instance: `by XBOOLE_1:15`.

The Mizar language is easy to read and write, due to the goals and aims of the project. Furthermore, we can state that the language itself is easy to learn. It was proved in many occasions and by many people trying to learn and understand Mizar. Furthermore, the Mizar language and its declarative approach is the base for development of a number of "modes", so called "Mizar Mode" for different procedural proof systems/checkers [Har96, GW03].

The difficult part of writing a Mizar article is to find existing notions and theorems as well as the right justifications and how to use and find references in the MML.

3.3 The Mizar Article

The Mizar *Article* is written as a text file which could be written in any editor. This provides great flexibility from the user point of view. The user is not restricted to any editor and can use his own preferable editor.

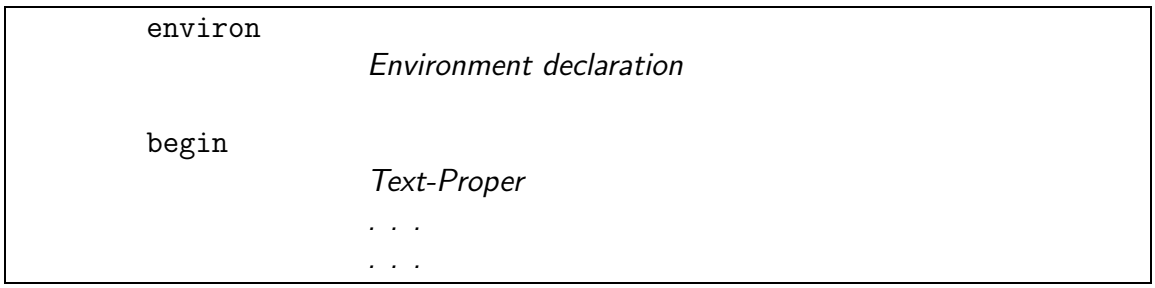


Figure 3.2: The Mizar article structure.

The Mizar *Articles* consists of two parts: the *Environment declaration* and the *Text-Proper*, as presented in Figure 3.2. The following two sections will aim to describe the structure of the Mizar article.

3.3.1 The Environment declaration

The *Environment-Declaration* begins with `environ` and consists of *Directives*, where each *Directive* 3.2 is composed of names of Mizar *Articles* imported from the MML. Each of these imported articles contains the knowledge required for verifying the correctness of the *Text-Proper*. Below is a short explanation of each *Directive* from the *Environment* [Ret05a]:

- *vocabularies:* This directive is used by the Mizar system for the lexical analysis of the *Text-Proper*. In the article one can only find reserved words and symbols that are given in the vocabulary files in this directive. The author can use the existing vocabularies and also is free to introduce a new one.

- *notations:* Articles placed in this directive are used to recognise the syntax of the *Text-Proper* part of the article. It contains articles in which the notations (definiendas) that are used in the *Text-Proper* part are defined.

- *constructors:* The articles written here are used by the Mizar system to interpret the meaning of the expressions in the *Text-Proper* part.

- *registrations:* Mizar has the possibility to cluster an adjective with an expression. For example the type "`set`" gets the adjective "`non empty`" in the

article XBOOLE_0. The *registrations* directive has to contain the articles in which clusters are defined that occur in the text proper.

- *requirements:* This directive contains articles the content of which is to be known automatically by the Mizar system during the processing of the article. Only five articles can be placed in this section: BOOLE, SUBSET, NUMERALS, REAL, ARITHM [NB04]

- *theorems:* The theorems directive holds the names of the MML articles that contain theorems and definitions that are used as references of reasoning steps.

- *definitions:* The definitions of predicates in the MML can be automatically unfolded. The names of the MML articles in which they occur have to be placed in this directive in order to recognise those definitions structure while verifying the article. This directive is mainly used if we use proofs by definition expansion.

- *schemes:* This directive contains names of the MML articles in which we introduced schemes that are used in the reasoning in the article. A *scheme* is a proposition that can take predicates as an argument, e.g., the mathematical Induction.

For instance, Listing 3.3 presents the Mizar *Environment* for the article regarding the properties of binary relations [Ret05b] formalised by K. Retel.

```
vocabularies RELSET_2, TARSKI, RELAT_1, CANTOR_1, SETFAM_1, BOOLE,
     FUNCT_1, PUA2MSS1, EQREL_1, FUNCT_5, SUBSET_1, COMPLEX1;
notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, SETFAM_1, RELAT_1,
     FUNCT_1, RELSET_1, FUNCT_2, EQREL_1;
constructors SETFAM_1, FUNCT_2, EQREL_1;
registrations XBOOLE_0, SUBSET_1, RELAT_1, PARTFUN1;
requirements SUBSET, BOOLE;
definitions XBOOLE_0, TARSKI, EQREL_1, SUBSET_1, RELAT_1;
theorems RELAT_1, TARSKI, SETFAM_1, ZFMISC_1, XBOOLE_1, SUBSET_1,
     RELSET_1, XBOOLE_0, FUNCT_1, FUNCT_2, MSSUBFAM, EQREL_1,
     SYSREL, ORDERS_1;
schemes FUNCT_1, DOMAIN_1;
```

Listing 3.3: The environment for the article regarding properties of Binary Relations [Ret05b].

The most difficult part of writing a Mizar article is identifying the existing knowledge in the MML and placing it in the proper *Directive* in the *Environment*

declaration. If we can't find in the MML the symbol or notation required for our theory, we have the possibility to create our new notation in the local vocabulary file. This local vocabulary file, with the extension "`.voc`", contains lines introducing new symbols, one line – one symbol. Each line begins with a capital letter which determines the kind of new symbol (see the Mizar homepage[3] and Table 3.3) followed by the name of the symbol. The constructor *functor* 3.5.3 may have additional information indicating the priority of the symbol, expressed as a number between 0 and 255, where the default is 64.

For instance, the following listing introduces a new adjective (annotated as `attr` in Mizar) for a binary relation, as seen in the source file `NECKLA_2.MIZ` of the Mizar article [Ret03b]:

```
definition
  let G be non empty RelStr;
  attr G is N-free means
  :Def1:
  not G embeds Necklace 4;
end;
```

The the vocabulary file `NECKLA_2.voc` has to contain the following line: `0N-free`, where `0` indicates the *attribute* constructor (i.e., adjective).

3.3.2 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*.

The current approach of the Mizar language does not support any meta-data assignment to a section, theorem or definition. However, Mizar users tend to write meta-data information as comments added after some important theorems or section. For instance, the following listing shows how we can provide some additional meta-data information to a section. The example is taken from [Ret05b] and provides a short description of the first section of the article.

```
begin :: Preliminaries

:: Formalisation of first paragraph from the article:
:: "Relations binaries, fermetures, correspondances de Galois" (1948),
```

[3] `http://mizar.uwb.edu.pl/language/vocabularies.html`

```
:: Prof. Jacques Riguet,
:: Bulettin de la S.M.F., tome 76 (1948), p.114-155.
...
```

The main work that needs to be done to encode a text in the Mizar language is to reveal the reasoning structure of a CML text. All terms, objects and their types used in the original document need to be identified and their MML counterpart simultaneously needs to be found. We encode the body of the original text in the main part – *Text-Proper* – of a Mizar document using the Mizar syntax and symbols found in the MML. At the same time we have to create the *Environment* which is an inseparable part of a Mizar article in the sense that we can't check, using the Mizar system, the correctness of the article without an *Environment*. The process of generating an appropriate *Environment* for a Mizar document consists in composing the *Directives* of the *Environment* with names of articles from the MML, that contain definitions of symbols, notations, constructors, theorems and definitions that are used in the main *Text-Proper* part of the Mizar encoding.

When proving a theorem in the Mizar language, the first thing to do is to check if the notions of that theorem are already formalised in the MML. If so, we need to include the articles from the MML in the appropriate *Directive* of the environment. If the notion is unknown to the Mizar system, the user has to adjust the environment. The notion has to be defined before stating the theorem in which this is used. The Mizar user has to also add the given name of this notion to the vocabulary file as well.

When the theorem is well written one can usually start with the proof. A line is well written, i.e., syntactically correct, in the Mizar language when the checker returns only justification errors: i.e., `*1` and `*4` errors, which mean: "`It is not true`" and "`This inference is not accepted`", respectively.

For example if we want to prove that $4 = \{0, 1, 2, 3\}$, the skeleton of the proof would look like as the one presented in Listing 3.4 If at this stage the *Environment* is created properly the Mizar checker should return only the `*4` errors. At this stage, the author has to remove the reasoning errors, and to provide full justification for each reasoning step. Listing 3.10 presents the fully formalised version of the above skeleton of the document and particularly the proof. The formalisation can now be processed by the Mizar system. If the Mizar system at this stage does not return any errors, we can be sure that the theorem is properly fully formalised.

```
theorem Th2:
  4 = {0,1,2,3}
proof
  set x = {i where i is Element of NAT: i < 4};
A1: x c= {0,1,2,3}
  proof
    let y be set;
    assume y in x;
    thus thesis;
  end;
A2: {0,1,2,3} c= x
  proof
    let y be set;
    assume y in {0,1,2,3};
    thus thesis;
  end;
  thus thesis;
end;
```

Listing 3.4: The skeleton of the theorem from the article `NECKLACE.MIZ` [Ret03a]

3.4 The Mizar System

In this section we will briefly describe the Mizar system and the processing of an article. We will show the processing and checking algorithm of an article based on the small example presented in Listing 3.4 from the previous section.

As already expressed the Mizar project consists of many parts developed at the same time. There are three core parts of the development: the Mizar language, the Mizar system and the Mizar Mathematical Library. The Mizar system is the only implementation of the Mizar Language, and its releases are distributed in all major operating systems[4].

The system consists of two major programs that process Mizar articles, these are: the *accommodator* and the *verifier*. As explained in the above section, the article is spread into two parts: *Enviornment-Declaration* and *Text-Proper*, which are processed by the *accommodator* and the *verifier*, respectively.

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*. If any of the above two programs return errors during the processing of a Mizar article, this information is flagged in the article by another program.

[4]MS Windows, Intel-based Linux, Solaris and FreeBSD, and also Darwin/Mac OS X and Linux on PowerPC – `http://mizar.uwb.edu.pl/system/`

Usually, the *accommodator* and *verifier* are called within the `mizf` user script, which is a bash script calling *acommodator*, *verifier* and error flag program in the proper order. Although, each of these programs can be called separately from the shell to process a Mizar article. Alternatively, one can use J. Urban's Emacs Mizar Mode[5] to write Mizar articles and verify them directly from the Emacs text editor. This Mizar Mode for Emacs provides a fully functional interface to the Mizar System.

3.4.1 Processing Mizar articles based on an example

Building a Mizar article is an iterative process of three steps:

- write theorems, definitions and reasoning steps,
- adjust environment,
- check/verify the already formalised document.

We describe a process of checking Mizar articles based on a small example from the previous Section 3.3.

Let us first write a simple theorem in the Mizar document named `MYTEST_0.miz`, as shown in Listing 3.5:

```
environ
begin

theorem
  4 = {0,1,2,3};
```

Listing 3.5: The theorem from the article [Ret03a] used as an example for describing the verification process of the Mizar article.

Then we check it using the command: `mizf text/mytest0.miz`.
As output we receive information, as shown in Listing 3.6, which indicates that there are two errors starting on the parser level.

The error place and description will be flagged in the Mizar article as expressed in Listing 3.7:

Basically those errors mean that there is not sufficient information provided to the Mizar system to be able to parse the text. In practice, this means that the environment has to be adjusted with the details of constructors, vocabularies

[5] `http://kti.mff.cuni.cz/~urban/`

```
Make Environment, Mizar Ver. 7.9.03 (Linux/FPC)
Copyright (c) 1990-2008 Association of Mizar Users

Verifier, Mizar Ver. 7.9.03 (Linux/FPC)
Copyright (c) 1990-2008 Association of Mizar Users
Processing: text/mytest0.miz

Parser    [    8 *2]    0:00
Analyzer  [    5 *2]    0:00
Checker   [    5 *2]    0:00
Time of mizaring: 0:00
```

Listing 3.6: The output of the verification of the example displayed in Listing 3.5

```
environ
begin

theorem
  4 = {0,1,2,3};
::>     *143    *152
::>
::> 143: No implicit qualification
::> 152: Unknown functor format
```

Listing 3.7: The output of the verification of the example flagged in the original document, as shown in Listing 3.5

and notations used in such statement. All of that information can be found in the MML by using different search engines, i.e., either using plain text search – `grep` or semantic search – MML Query[6] (developed by G. Bancerek). A short description is given in Section 3.5.2.

Once we adjust the environment we should receive only one error `*4` which states that the statement is not sufficiently justified.

```
environ
 vocabularies ARYTM;
 notations ENUMSET1, NUMBERS;
 constructors REALSET1;
 requirements BOOLE, SUBSET;
begin

theorem
  4 = {0,1,2,3};
::>           *4
::> 4: This inference is not accepted
```

[6] `http://merak.pb.bialystok.pl/mmlquery/three.html`

Running the `verifier` or the `mizf` on the above Mizar article will result in the output presented in Listing 3.8:

```
Make Environment, Mizar Ver. 7.9.03 (Linux/FPC)
Copyright (c) 1990-2008 Association of Mizar Users

Verifier, Mizar Ver. 7.9.03 (Linux/FPC)
Copyright (c) 1990-2008 Association of Mizar Users
Processing: text/mytest0.miz

Parser    [  11]    0:00
Analyzer  [   9]    0:00
Checker   [   9 *1]    0:00
Time of mizaring: 0:00
```

Listing 3.8: The output of the verification of the example displayed in Listing 3.5, containing only one justification error.

Now we can build the skeleton of the proof, as expressed in Listing 3.4, and adjust the environment.

And finally we provide justification of each reasoning step in the document. This is done by (1) finding proper theorems/definitions from the MML that are required to prove a statement, (2) placing the name of those articles that contain theorems/definitions which are useful for the justification inside the *Enviornment-declaration*, and (3) further adjustment of the *environment*.

The final version of the example is presented in Listing 3.9.

3.5 The Mizar Mathematical Library

The original goal of the Mizar project was to design a language and implement a software assistant to support writing traditional mathematical papers. The ongoing experiments with the Mizar system formed the main trend of its development. Moreover, it became the main goal to build a centrally maintained library of mathematical documents that are verified by the Mizar system. In 1989 the Mizar group has started a project of building the Mizar Mathematical Library (MML). The MML is an ever-growing collection of formal mathematics written in the Mizar Language and based on the Tarski-Grothendieck set theory[Ban90b].

At the moment for around two decades, the MML is the largest, all over the world, library of formalised and computer checked mathematics. It consists of 1011 Mizar articles contributed by more than 209 authors, containing 46506 theorems

```
environ
 vocabularies ARYTM;
 notations TARSKI, ENUMSET1, SUBSET_1, NUMBERS, XXREAL_0;
 constructors SQUARE_1, NAT_1, REALSET1;
 registrations ORDINAL1, XREAL_0;
 requirements BOOLE, SUBSET, NUMERALS, ARITHM;
 definitions TARSKI;
begin

theorem Th2:
  4 = {0,1,2,3}
proof
  set x = {i where i is Element of NAT: i < 4};
A1: x c= {0,1,2,3}
  proof
    let y be set;
    assume y in x;
    thus thesis;
::>            *4
  end;
A3: {0,1,2,3} c= x
  proof
    let y be set;
    assume y in {0,1,2,3};
    thus thesis;
::>            *4
  end;
  thus thesis;
::>          *4
end;
::> 4: This inference is not accepted
```

Listing 3.9: The sketch of the proof of the example given in Listing 3.5.

```
environ
 vocabularies ARYTM;
 notations TARSKI, XBOOLE_0, ENUMSET1, SUBSET_1, NUMBERS,
   XCMPLX_0, XXREAL_0;
 constructors REAL_1, SQUARE_1, NAT_1, REALSET1, WAYBEL_1;
 registrations XBOOLE_0, ORDINAL1, XREAL_0;
 requirements BOOLE, SUBSET, NUMERALS, REAL, ARITHM;
 definitions TARSKI,SUBSET_1;
 theorems NAT_1, ENUMSET1, AXIOMS, XBOOLE_0;

begin :: Preliminaries

theorem Th2:
  4 = {0,1,2,3}
proof
  set x = {i where i is Element of NAT: i < 4};
A1: x c= {0,1,2,3}
  proof
    let y be set;
    assume y in x;
    then consider i being Element of NAT such that
A2: y=i & i < 3+1;
    i <= 3 by A2,NAT_1:13;
    then i = 0 or i=1 or i=2 or i=3 by NAT_1:28;
    hence y in {0,1,2,3} by A2,ENUMSET1:def 2;
  end;
A3: {0,1,2,3} c= x
  proof
    let y be set;
    assume y in {0,1,2,3};
    then y=0 or y=1 or y=2 or y=3 by ENUMSET1:def 2;
    hence y in x;
  end;
  thus 4 = x by AXIOMS:30
    .= {0,1,2,3} by A1,A3,XBOOLE_0:def 10;
end;
```

Listing 3.10: The full formalisation of the example given in Listing 3.9.

and 8804 definitions[7].

An article, when published into the MML, gets a unique identifier (name), which is rather meaningful and indicates its contents. For instance the `CARD_1.MIZ`[Ban90a] article introduces and states properties of cardinal numbers. Furthermore, each article when published is processed into several forms and is distributed with the Mizar software:. This includes:

- the source text of the article – the `.miz` file – containing all theorems (local and public) with their proofs,
- the *abstract* of the article – the `.abs` file – containing public information, i.e., only definitions and theorems that were labeled in the `.miz` file and used in the justification of other theorems,
- database files – used when importing items from the article to a new article environment.

The MML forms the basis of every new Mizar article. Every concept and notation that is in the MML can be reused in a new article written. Furthermore, theorems and definitions stored in MML are used in justification for reasoning steps of new theorems.

The Mizar group has begun a project for building an Encyclopedia of Mathematics in Mizar (EMM). At the moment, there are 10 articles forming the EMM[8]. Those articles have mono-graphical character and are extracted from the `.miz` files of contributed articles.

3.5.1 Complex theorems and books formalisation.

A wide range of the MML consists of theorems and definitions of more or less complex mathematical problems. However, there exists also very complex theories and theorems whose formalisation was labor intensive and demanded lots of effort.

The more important facts included in the MML, that are worthwhile mentioning, are [9]:

- the Jordan Curve Theorem,

[7] version 4.100.1011 of the MML

[8] Series of articles which name is prefixed with letter "X", i.e., `XBOOLE_0`, `XREAL_0`, etc.,. Visit the url `ftp://mizar.uwb.edu.pl/pub/version/doc/mml.txt` and search for the EMM acronym; last time viewed the page on August 21, 2008.

[9] `http://merak.pb.bialystok.pl/` and `http://www.cs.ru.nl/~freek/100/`

- the Brouwer Fixed Point Theorem,
- the Gödel Completeness Theorem,
- the Fundamental Theorem of Algebra,
- the Reflection Theorem,
- A Small Fermat's Theorem,
- the Fundamental Theorem of Arithmetic, etc.,

The formalisation of these theorems resulted in a number of Mizar articles deployed to the library, whose knowledge is heavily reused in other articles.

At the 2nd QED[10] Workshop held in Warsaw in 1995 the following question was raised:

> Can we do formalisation of advanced mathematics like this included in regular mathematical monographs in the current proof-checking system? *[RT99, pp.2]*

At that time the Mizar Group has followed "Ralph Wachter's suggestion to put Mizar under a stress test by starting the formalisation of a *A Compendium of Continuous Lattices* [GHK+80] in its entirety" (cited from [BR02]). The project officially started in April 1996, and involved team effort of more than 16 participants mainly from University of Bialystok with some contributions from members in Canada and Japan [Ban00, BR02].

The formalisation of the CCL-book[11] resulted in a number of Mizar articles which hugely strengthened the MML development and its current growth.

The CCL project formalised slightly above 60% of the entire CCL-book. The book contains 334 pages covering 715 items (definitions, theorems, exercises, etc.), on which 254 are examples and exercises which were not intended to be formalised during the project time. By the end of April 2002, the project covered around 231 items [BR02].

The formalisation of the CCL-book resulted in two series of articles with identifiers prefixed by: `YELLOW`[12] and `WAYBEL`[13]. The former series of articles (`YELLOW`) was aimed to bridge the gap between existing content of the MML (at the time

[10] `www-unix.mcs.anl.gov/qed`
[11] We refer to "*A Compendium of Continuous Lattices*" using acronyms: CCL or CCL-book.
[12] Nobody remembers the origin of the name.
[13] It is related to the main concept of *way below* in continuous lattices.

of the running project) and the knowledge assumed in the CCL-book. The latter series of articles, is the formalisation of the main course of the CCL-book.

	MML	WAYBEL	YELLOW	Percentage
Articles	717	35	22	7.95%
Theorems	31741	1512	1018	7.97%
Definitions	6093	271	138	6.7%
Schemes*	756	55	20	

Table 3.2: Some statistics of the CCL-book formalisation

The CCL project showed that the Mizar language is expressive enough to be able to handle the formalisation of an advanced and quite recent mathematics. Although, the project also shows that a decent library of formalised mathematics has to exist prior to the formalisation of further more advanced mathematics. In the CCL-book formalisation quite a decent number of references to external, previously formalised theories, were used. This also shows that at the time of the beginning of the CCL project, the MML was big enough to proceed.

A full project report regarding the CCL-book formalisation could be found in [BR02] written by G. Bancerek and P. Rudnicki. Statistics presented in Table 3.2 are taken from [BR02] and are based on the MML version containing 717 Articles.

3.5.2 The MML Query

Due to the large size and structure of the MML, searching over the whole library is not an easy task. The search of a theorem can be done in various ways:

1. by using any plain text search command – e.g., `grep`,
2. by looking at vocabulary files for certain definitions and by opening the corresponding articles,
3. by clicking on notions within the Emacs Mizar Mode written by J. Urban,
4. by using G. Bancerek's query tool – the MML Query.

The first 3 ways of searching MML (1–3) are rather easy and straightforward, if the user has a rough idea of the content of the MML. These three ways of search are syntax oriented rather than semantic oriented. Furthermore, the result for a query performed using these first 3 types of search could be less accurate than any semantic search performed. Moreover, when searching MML for a notion and

constructor of a symbol, the search could be more troublesome and sometimes might be even impossible, than if searching for a theorem. Therefore, G. Bancerek has been developing the semantic search tool called MML Query. It is the only semantic search engine implemented for the MML.

The MML Query builds its own database from the MML by extracting all available symbols, notations, formats, patterns, constructors, theorems, definitions etc. It has its own format of storing and naming files.

When trying to retrieve vital information from the MML it is important to understand some constructors and notations used in the MML and MML Query. A Mizar definition defines a new constructor and gives a syntax and its meaning. For example, Listing 3.11 introduces the constructor *functor*.

```
definition
  let X,Y be set;
  func X \/ Y -> set means
:: XBOOLE_0:def 2

  x in it iff x in X or x in Y;
  commutativity;
  idempotence;
end;
```

Listing 3.11: The Mizar *functor* constructor definition.

The *format* of a constructor specifies the symbol (i.e., \/ in our case) of the constructor and the place (i.e., infix position) and the number of arguments (i.e., 2 arguments). The format of a constructor together with the information about the types of arguments is called the *pattern*. The *formats* are used for parsing and the *patterns* for identifying constructors. Table 3.3 presents a short overview of available constructors in the Mizar Language and MML.

Constructor	*Syntactic construction*	*Constructor kind/code*	*Notation kind/code*	*Vocabulary symbol tag*
`Aggregate`	Structural term	`aggr`	`aggrnot`	`G`
`Attribute`	Adjective	`attr`	`attrnote`	`V`
`Functor`	Term	`func`	`funcnot`	`O`
`Mode`	Type	`mode`	`modenot`	`M`
`Predicate`	Atomic formula	pred	`prednot`	`R`
`Selector`	Structure selector	`sel`	`selnot`	`U`
`Structure`	Structure type	`struct`	`structnot`	`G`

Table 3.3: Mizar Language constructors and notations.
The original description of Mizar constructors and MML Query is published in [BR03]. The table is adopted from [BR03].

A simple query of the MML Query displaying all the constructors from the NECKLACE.MIZ article could be written as follows: `list of constr from NECKLACE;`.

If we run this query in the MML Query web application[14] we will receive the result as shown in Figure 3.3:

```
MML Query, version 1.4.01, MML 4.100.1011

QUERY: list of constr from NECKLACE 10 element(s)
NECKLACE:attr 1 => symmetric;
NECKLACE:attr 2 => asymmetric;
NECKLACE:attr 3 => irreflexive;
NECKLACE:func 1 => -SuccRelStr;
NECKLACE:func 2 => SymRelStr;
NECKLACE:func 3 => ComplRelStr;
NECKLACE:func 4 => Necklace;
NECKLACE:pred 1 => embeds;
NECKLACE:pred 2 => embeds;
NECKLACE:pred 3 => is_equimorphic_to;
```

Figure 3.3: The MML Query result for a simple query.

Summarising, MML Query is a semantic search engine allowing to retrieve information from the MML by using meanings of Mizar constructors. Although it is quite tedious tool to use, it is worthwhile trying and experimenting. The number of results that we get from queries applied to the MML Query are far much more narrowed than any other search command.

An advantage of using MML Query is while building a Mizar environment for a new article. The MML Query provides a simple hint of names of articles that have to be placed within directives in the *Environment-declaration* part. This is available within the online version of the MML Query [15].

Further description of the MML Query as well as a number of small examples of its usage could be found in [BR03] or in the webpage[16].

3.5.3 Mizar types

The Mizar language provides number of constructions to define new constructors. Those constructions were presented in the previous section. Among these constructions there are two concepts that introduce types in Mizar:

[14] http://merak.pb.bialystok.pl/mmlquery/three.html
[15] http://merak.pb.bialystok.pl/mmlquery/three.html
[16] http://merak.pb.bialystok.pl/mmlquery/three.html

radix-type	the construction	*adjective*	the construction
`mode`	the constructor	`attr`	the constructor

In the MML we distinguish types without arguments, like:
`set`, `non empty set`, `reflexive transitive Relation`, `N-free`, `Group`, etc.,
and types with a non empty list of arguments like:
`Element of A`, `Relation of A,B`, `normal Subgroup of G`, etc.,
where `A` and `B` are of type `set` and `G` is of type `Group`.
It is important to note that Mizar types consist of two parts: a list of adjectives (possibly empty), e.g., `non empty` and `reflexive transitive`, and a radix-type, e.g., `set` and `Relation` in the above examples.

The widest type in Mizar is called `set` – any Mizar type is a subtype of type `set`. It is introduced in the article `HIDDEN`, where it contains definitions of primitives (introduced without a definiens): type `set`, predicate `=` and predicate `in`. Those primitives are built-in notions of Mizar articles [Ban03]. The article is added to the environment of any new article automatically by the Mizar system.

```
definition
  mode set;
end;
```

Now we can introduce the *adjective* `empty` using the construction ***attr***, as shown in Listing 3.12:

```
definition
  let X be set;
  attr X is empty means
:: XBOOLE_0:def 5
  X = {};
end;
```

Listing 3.12: The Mizar ***attr*** constructor definition. The example is taken from article `XBOOLE_0` [Com02].

As we see, *radix-types* and *adjectives* depend on terms only. Therefore, we may say that we get *radix-type* by applying `mode` to a list of terms [Ban03]. In the above listing we register term `x` with type `set`. An *adjective* is obtained by the application of attributes to a list of terms extracted from the *radix-type* [Ban03].

If we want to apply the adjective `empty` to the type `set` we need to prove the cluster `empty set` which states the existence of a set which is empty.

```
registration
  cluster {} -> empty;
  cluster empty set;
end;
```

Once this registration cluster is proved we can use the `empty set` as a legal Mizar type, so we can write an expression: `let x be empty set`, which will be understood by the Mizar system.

Another type of `mode` definition might be used to provide a mother type of a newly introduced type.

```
definition
 let x be set;
 mode subset of x -> set means it c= x;
 existence proof ... end;
end;
```

For example, the above listing (taken from [Ban03], not existing in the MML) introduces a new type `subset of` which widens to type `set`. In other words if something is `subset of x` (where `x` is set) it is also a `set`.

It is also important to note that we can introduce new types by using the concept of structure:

structure-type	the construction
`struct`	the constructor

The first basic structure-type defined in the MML is introduced as follows:

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

This definition introduces type `1-sorted`, which has one field called `carrier` of type `set`. This structure could be later on used as an ancestor in other definition, which extend the mother structure-type:

```
definition
  struct(1-sorted) RelStr (# carrier -> set,
    InternalRel -> Relation of the carrier #);
end;
```

Listing 3.13: The *Binary Relation* definition. The example is taken from article `ORDERS_2.MIZ` [TB90].

The above listing introduces a new structure-type `RelStr`. This structure type is an ancestor for quasi ordered sets, posets, semi-lattices and lattices etc. For the `RelStr` type we can define new adjectives and apply them to modify the original type and finally to create a new type.

```
definition
  let A be RelStr;
  attr A is reflexive means
:: ORDERS_2:def 4
  the InternalRel of A is_reflexive_in the carrier of A;
end;
```

Then we prove an existence of such `reflexive RelStr`. Similarly, we can define the adjectives `transitive, antisymmetric` for type `RelStr`. And finally we can introduce a new type for *partial order set* – which consists of a set X and a binary relation R over that set which is `reflexive, transitive and antisymmetric`.

```
definition
  mode Poset is reflexive transitive antisymmetric RelStr;
end;
```

It is also important to note, that Mizar allows *overloading*, which means that the same symbol could have several meanings. For instance the listings presented in Figure 3.4 introduce the same *adjective* `reflexive` which is applied to a term of different type.

Furthermore, Mizar has another well known feature – *polymorphism*. This means that one *functor* could have different result types for different types of arguments.

Those features, *overloading* and *polymorphism*, are not specific to Mizar. They have been found very useful in modern programming languages.

3.5.4 Formalized Mathematics

As said before, the Mizar project consists and contributes with several things as stated in Section 3.1. Among those things is Formalized Mathematics.

The Formalized Mathematics (FM) is a quarterly published journal of mathematical papers checked by the Mizar system. It is a paper edition of abstracts of Mizar articles contributed to the Mizar Mathematical Library. The FM is published by the University of Bialystok, where the role of Editor-in-Chief is held by R. Matuszewski and the role of Scientific Editor is held by G. Bancerek.

```
definition
  let C be AltGraph;
  attr C is reflexive means
:: ALTCAT_2:def 6
  for x being set st x in the carrier of C
  holds (the Arrows of C).(x,x) <> {};
end;
```

```
reserve i, x, I for set,
  A, M for ManySortedSet of I, [...]

definition
  let I, M;
  let IT be MSSetOp of M;
  canceled;
  attr IT is reflexive means
:: CLOSURE1:def 2
  for X being Element of bool M holds X c= IT..X;
end;
```

Figure 3.4: The Mizar *overloading* example. The same attribute (*attr*) has given two different meanings.

The Mizar article that is aimed to be submitted to the MML and at the same time to the FM has to comply few things:

1. has to be verified, without any errors, by the latest release of the Mizar system,

2. the content of the article should be original and interesting, which means that definitions and theorems presented in the new article are not part of the library yet,

3. the article has to pass the reviewing process.

The reviewing process, described in [GS07], has been introduced and follows commonly used scheme accept/revise/reject. All papers are reviewed by at least three experts from the relevant field [17].

All papers submitted to the FM are checked by the Mizar system and automatically translated from the Mizar Language to the LaTeX typesetting format. Once the LaTeX format is processed we get the Mizar document written in English which mimics the CML. The system for automatic translation and typesetting with LaTeX has been designed and developed by G. Bancerek. The design of the system is based on the previous works of A. Trybulec and Cz. Byliński.

[17] http://fm.mizar.org/about.htm

3.6 Some of Our Formalisations in Mizar

In this section we present a short description of formalisation attempts carried out by this PhD student K. Retel. Such formalisations carried out by this student allowed to gain inner understanding of the Mizar system. In addition, these formalisations enhanced this student's knowledge, which is re-used in the current MathLang development.

Mathematics nowadays is very complex and requires a wide variety of prior knowledge to truly understand mathematical facts presented in a journal paper. Furthermore, recent mathematical papers are rarely self-contained in terms of the knowledge presented in these articles. Therefore, the formalisation of a mathematical document might be very labour intensive. In addition to that, most of the current approaches to the formalisation of mathematics have as a main goal to formalise recent mathematics and to test the suitability of the formalisation language and of the system in which the formalisation is planned to be performed.

Therefore before the formalisation of a recent mathematical paper, there should exist a big database of mathematics already formalised and verified by the computer. Such a database should provide a number of definitions and theorems that could be reused in the formalisation of new mathematical facts. Such a library of prior formalised mathematical knowledge has lots of advantages for potential proof system users. Of these advantages we mention the provision of case studies for users making the formalisation of new facts much easier and less labour intensive. For instance, instead of formalising some definitions required prior to the formalisation of a theorem, we can reuse an already carried out formalisation of these definitions if such formalisation exist already in the database. Furthermore, in the same manner, as the reader requires prior knowledge to understand new fact, similarly a computer proof system requires previous theorems and definitions to fully justify new facts and their proofs to be formalised. Moreover, the existence of such a database saves time of the formalisation of new facts. It has been widely recognised that developing a library on which further formalisations can be based, is essential.

The Mizar system is accompanied with such a library which is called the Mizar Mathematical Library (MML). The MML comprise about 50,000 theorems and 10,000 definitions and other items from a wide range of mathematical domains.

3.6.1 Formalisation of finite series-parallel graphs

The first formalisation attempt was to formalise a relatively recent mathematical paper published in a well known mathematical journal. In 2001, Dr A. Trybulec has chosen a document (published in year 2000) regarding binary relations, graphs, trees, necklaces and well-ordered relations. One of the reasons for such a choice was, that the MML, at the time of starting this formalisation project, contained enough mathematical facts on which the formalisation could be based. Having this in mind, it was estimated that the formalisation project should not take long especially if the library contained a number of facts that could be reused. However, at that time, the student K. Retel had very little expertise regarding Mizar and the MML.

The original document [Tho00] chosen for the formalisation was written by S. Thomasse and was published in the journal of "Transactions of the American Mathematical Society" in 2000.
The actual formalisation project started in late 2002. It had number of different goals, among which there were the following one:

- test the Mizar system and its library,
- prove that Mizar is mature and strong enough to formalise recent mathematics,
- prove that the Mizar Mathematical Library comprises enough mathematical knowledge already formalised which provide a strong basis for further formalisation of relatively recent mathematics,
- familiarise the student (K. Retel) with the Mizar system and the MML, so that he can start working within the Mizar group.

This formalisation project has resulted in a series of 3 Mizar articles (named: `NECKLACE.MIZ`, `NECKLA_2.MIZ`, `NECKLA_3.MIZ`, see Appendix C) that were submitted to the MML and published in the Formalized Mathematics journal.

These three articles provide an extension to the MML. A number of new facts (definitions, theorems and lemmas) has been formalised. These facts were not present in the MML before starting the project. In addition to that, a number of these new facts were omitted by the author in the original journal paper or assumed to be known by a reader who wanted to study such paper. It is recognised as a common practice in mathematical world that mathematicians omit a number

of "trivial" facts/proofs and assume that a reader possesses enough mathematical knowledge to fully understand a paper. However, proof systems require some of those facts to be explicitly formalised.

For instance, the first theorem proven within the first article of the necklace's series regards to the enumeration set, mainly: "$4 = 0, 1, 2, 3$". This enumeration set is used within the original S. Thomasse article and is related to the definition of the $N - free$ graph, which will be explained in small details later on.

The series of three Necklace articles contain a formalisation of only the first and second section of the original S. Thomasse's article. Within these two sections we can find a number of facts related to binary relations. The first section, titled "The class of series-parallel graphs" begins with an introduction of binary relations. This definition was already formalised in Mizar, as shown on the previous Listing 3.13, therefore we reused this fact.

Another definition, after the definition of the binary relation, specifies the embeding relation between two binary relations which in the original document is written as follows:

> "A binary relation $R = (V, E)$ is *embedded* in another binary relation $S = (W, F)$ (or S *embeds* R) if there is an injective mapping f from V into W such that $(x, y) \in E$ if and only if $(f(x), f(y)) \in F$."

This definition is introduced in the first Mizar article `NECKLACE.MIZ` as shown on listing 3.14.

```
definition
  let R,S be RelStr;

  pred S embeds R means
:: NECKLACE:def 2

  ex f being Function of R,S st f is one-to-one
  & for x,y being Element of R holds
  [x,y] in the InternalRel of R iff
  [f.x,f.y] in the InternalRel of S;
end;
```

Listing 3.14: The Mizar formalisation of the *embedding* definition between two binary relations.

Going further the author S. Thomasse introduces a definition of a *graph*:

> "A graph $G = (V, E)$ is an irreflexive and symmetric binary relation"

The enumeration set 4, as shown at the beginning of this section, is used in the definition of the N graph, as denoted by the S. Thomasse in the original paper.

> "We denote by N the graph with vertex set $\{1, 2, 3, 4\}$ and edge set $\{(1, 2), (2, 3), (3, 4)\}$."

It is easy to note that this definition is not complete, and some information is left to the reader. Mainly, S. Thomasse introduces a *graph* as a symmetric binary relation. This means that the set of edges of an N graph should also contain symmetric edges. Therefore the formalisation of such graph looks like:

```
for n being Nat holds
 the carrier of Necklace 4 = {0,1,2,3} and
 the InternalRel of Necklace 4 =
                              {[0,1],[1,0],[1,2],[2,1],[2,3],[3,2]};
```

Listing 3.15: The Mizar formalisation of the N graph..

For readability and brevity, we present here a compilation of different definitions and theorems from the articles: `NECKLACE.MIZ` *and* `NECKLA_2.MIZ`*. The reader can find these definitions and theorems in Appendix C.1*

The shape of such an N graph when sketched on a piece of paper resembles a *necklace*. Hence we decided to give the name $Necklace$ to the definition of the N graph. Similarly names of the articles have a word $Necklace$. One can argue that the name of these Mizar articles should be different and more self-explanatory. However, this is left to an author decision and can be advised to change by the Library Committee[18] and Referees [GS07].

Another definition presented in the original paper states that:

> "A graph is $N - free$ if it does not embed N."

It has been formalised in Mizar as follows:

```
definition
  let G be non empty RelStr;
  attr G is N-free means
:: NECKLA_2:def 1
```

[18] A. Grabowski is the head of the Mizar Mathematical Library committee (`http://www.mizar.org/library/committee.html`) and is responsible for MML revisions and MML production.

```
  not G embeds Necklace 4;
end;
```

The second article from the Mizar series of Necklaces is a continuation of the formalisation of the first section of the original paper. It covers mainly the formalisation of one theorem stated in the first section of the S. Thomasse paper:

> **Lemma 1.** (Gallai [4]) *The class of finite series-parallel graphs is the class of finite $N - free$ graphs.*

The formulation of such a theorem is expressed as one line in the NECKLA_2.MIZ article but involves the notation of a definition introduced earlier in the Mizar article.

```
theorem :: NECKLA_2:7
  for R being strict non empty RelStr st R in fin_RelStr_sp
  holds R is N-free;
```

The actual formalisation of the proof of this theorem was very labour intensive and refers to facts that the student K. Retel had to introduce before that theorem and on which the proof is based. The original proof, in the journal paper reproduced by S. Thomasse, occupied 11 lines on an A4 sheet of paper (approximately one fourth of the A4 sheet of paper). Whereas its formalisation covers 1407 lines and consumes approx. 52 KBytes of disk memory [19].
It is not a surprise that the formalisation of the theorem is quite big compared to its LaTeX informal presentation. It was found by de Bruijn that the factor expressing the size of the formalisation of a theorem compared to its original informal presentation in LaTeX may be quite big but that it is rather constant, i.e., it does not increase in further formalisations of a mathematical theory.

The last Mizar article NECKLA_3.MIZ (see Listing C in Appendix C.3) contains the formalisation of a number of facts relating to two operations defined in the original document, mainly the union and the sum of two graphs. These definitions were introduced within the previous Mizar article: NECKLA_2.MIZ (see definitions NECKLA_2:def 2 and NECKLA_2:def 3 in the Listing C.2 in Appendix C.1). The author S. Thomasse in the original document introduces as synonyms for these operations the following: series and parallel for union and sum operations, respectively. In the Mizar formalisation, K. Retel didn't introduce these synonyms and only used the following one: *union_of* and *sum_of*. The reason for that is that a

[19] In comparison to that the whole NECKLA_2 covers 1980 lines and approx. 72KByte of disk memory.

Mizar user carrying out some formalisation has a complete freedom in terms of providing names, structures and proofs to each definition, theorem and proof in the formalisation. Moreover, the user can not be forced to change provided names to definitions but might be advised by referees or the Library Committee [GS07].

The formalisation of these definitions was enough to formalise the proof of the main theorem within the first section of the original article. `NECKLA_3.MIZ` article introduces a number of theorems and proves properties that relate to these two operations. Some of these facts were not even mentioned in the original S. Thomasse's document.

It is important to remember that the formalisation of an informal paper is left for an author who is actually doing the formalisation. Therefore the structure of a formalisation of an informal document can be laid out in any possible way. Moreover, the author formalising an informal document has full control over his formalisation. Of course there are some rules that the author has to follow in case he wants to submit his formalisation into the MML. However, names given to definitions, as well as the structure of a file and the formalisation style are left to the author taste and experience.

3.6.2 Formalisation of some binary relations properties

The second attempt of formalisation in Mizar was to complement the knowledge contained in Mizar Mathematical Library regarding binary relations. Similarly to the previous formalisation attempts, the next article to formalise was chosen by A. Trybulec.

The original document [Rig48] chosen for formalisation was written by J. Riguet and was published in "Bulletin de la Societe Mathematique de France" in 1948. The actual formalisation project was done by K. Retel in March – April in 2005, and covers proofs of all theorems and properties stated in the first section of the original document [Rig48]. Some of these facts were already formalised and submitted to the MML. Numbers of facts that were already formalised before the formalisation project started, as well as their corresponding formalisation are mentioned within the Mizar article (see Listing C.4 in Appendix C.2).

Similarly to the previous formalisation attempt this attempt has been published and included in the current distribution of the MML. The name of the file directly correspond to the domain which this file is attempting to formalise, mainly it is `RELSET_2.MIZ`, a second file from the series of binary relations. All the theorems from that formalisation file are based on and take into account the binary relations.

Therefore we reused the already formalised facts and definitions of binary relations, as presented in Listing 3.13 in the first article `RESLET_1`.

In `RELSET_2.MIZ`[Ret05b], K. Retel defines an image and an inverse image of an element of set A under a binary relation of two sets A, B, as an image and an inverse image of a singleton of the elements under this relation, respectively. Next, he introduces "The First Order Cutting Relation of two sets A, B under a subset of the set A" as the union of images of elements of this subset under the relation, see `RELSET_2:17` in Appendix C.2. Furthermore, K. Retel formalises "The Second Order Cutting Subset of the Cartesian Product of two sets A, B under a subset of the set A" as an intersection of images of elements of this subset under the subset of the Cartesian product, see `RELSET_2:def 4` in Listing C.4 in Appendix C.2.

The formalisation also defines the first and second projection of binary relations, see `RELSET_2:51` in Listing C.4, Appendix C.2. The main goal of the formalisation of the original article is to prove properties and collocations of introduced definitions in the original informal paper. The numbers written in parenthesis after the label of theorems in the formalised article `RELSET_2.MIZ` correspond to the numbers of expressions contained in the original article.

This attempt at formalisation was less labour intensive than the first attempt described in the previous section. Nonetheless, it involves a lot of knowledge regarding the MML and formalisation techniques.

Summarising, expertise gained from these formalisation attempts allowed K. Retel to understand the Mizar way of representation and formalisation styles. Moreover it allowed him to understand the processes involved during checking an article using the Mizar system. All articles formalised during the attempts are included in this thesis as appendices.

3.7 Mizar as a Tool for Teaching Mathematics

This section provides a short description of projects in which the Mizar system was used as a computer assistance tool for teaching mathematics. This PhD student has been actively involved in some of them which resulted in a short technical report published in a special issue of a workshop proceedings [RZ05] dedicated to the event of "30 Years of Mizar"[20].

The development of the Mizar system is conducted by many experiments. Some

[20]The workshop was held during the MKM2004 conference, and it celebrated, as the name suggests, 30 Years of Mizar.

of these experiments concerned education and especially the teaching of mathematics with computer assistance.

In the past, many experiments of the teaching of mathematics with use of the Mizar system as a computer assistance were held [RZ05]. In this section, we present them in a short list detailing the date and the subject of the taught courses.

1975–1976 – the first use of Mizar as a tool for teaching mathematics. At that time, the first implementation of the Mizar processor was used to teach *propositional logic*.

1983 – 1984 (September through June) – an interesting correspondence course based on Mizar-MSE was run for 10 months by the Polish popular science monthly *Delta*, a magazine aimed at secondary school students. The course was aimed to teach *an introduction to logic* using Mizar-MSE.

1985–1986 – the same Mizar version (Mizar-MSE) was used to teach *elementary logic* in a number of one semester courses.

spring 1985 – a course in *foundations of geometry* was taught. The course was based on a textbook [KS76] covering the formal exposition of axiomatic Euclidean geometry.

1987–1988 – a much richer version of Mizar, called Mizar-4, was applied in teaching *introduction to mathematics* and *lattice theory*.

early 1990's – PC-Mizar was "indirectly" used in teaching *topology*.

90's – Mizar was used for teaching *introductory logic* courses at many universities in countries like Canada, USA, Japan and Belgium.

after 1997 – Mizar was used for teaching different courses for students at university level. Even, this PhD student at 1997–1998 was taught an introduction to logic using Mizar as a computer assistant.

2003–2007 – Mizar was applied for teaching two courses: (1) *Introduction to Logic and Set Theory*, and (2) *Formalisation of Mathematics*. Both courses were obligatory part of the curriculum for all first-year students of the Institute of Computer Science, University of Bialystok [RZ05, BZ07].

The above courses were taught mainly at universities, and were aimed at all levels from the first-year students to the final-year students.

The next couple of paragraphs will present an overview of the course of *Formalisation of Mathematics* that was conducted in the academic year 2003–2004. At that time, K. Retel had been involved in preparing material, tutoring students, writing exams and marking students progress. The material presented here is given in more details in [RZ05].

The main goals of the course were as follows:

- extending students' deduction skills by selecting exercises, within the scope of the preselected areas of mathematics, requiring the usage of more advanced proof techniques,
- teaching a selected mathematical theory presented during the lecture,
- presenting the range of techniques used in the formalization of mathematics.

To meet these objectives and the university curriculum the theory of *binary relations* has been chosen as the basis of the course. The chosen theory was proved to be sufficiently rich even when material for the exercises was notably restricted. We[21] have decided that the use of the whole MML would be too complicated and time consuming for students. Therefore, we created 5 new Mizar articles that meet lectures materials. Although, these new environments were created, their content was taken directly from the MML and included: unordered pairs, relations, selected properties of relations, relations on two sets, cartesian product, singletons, ordered pairs, operations on sets, union, intersection and power of sets, scheme of mathematical induction.

During the course the total number of 140 exercises (theorem statements that required a proof from a student) were created. However an average student proved 50 theorems.

This course proved again that a *proof system* can be used as a tool for teaching mathematics. Instead of teaching students how to prove using pen & paper and of being involved in the process of checking a human proof (usually a tutor), it is worthwhile to involve a computer *proof system*, especially in the millenium of computers and where the computer can be an assistant in teaching.

The course has proved for another time, that the Mizar system is highly suitable for teaching mathematics. The reason for this is that the Mizar language is expressive and easy to learn, to read and write even for first-year students. It also

[21] In the following part of this section the word "we" stands for the group of people involved in organising and tutoring this course, i.e., A. Trybulec, A. Naumowicz, K. Retel, A. Rybak and A. Zalewska

showed that main natural deduction techniques[22] could be taught with success. Using Mizar showed that students are getting more familiar, from their early years at university, with more formal methods and especially with formalisation techniques.

3.8 The Mizar FPS

In this section, first we describe the Formal Proof Sketch (FPS) notation that has been invented and introduced by F. Wiedijk. He introduced FPS to any declarative proof language and presented the whole description of it in a number of publications [Wie04a, Wie04b].

3.8.1 The Formal Proof Sketch (FPS)

The Formal Proof Sketch notion was introduced by F. Wiedijk in [Wie04a] 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 F. 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 formalization should be *justification errors*: the errors that say that a step is not sufficiently justified by the references to previous steps. *[Wie04a]*

Even if the above definition states that the FPS version is derived from full formalisation, 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 formalisation.

3.8.2 The Mizar Formal Proof Sketch (Mizar FPS)

The Mizar Formal Proof Sketch (Mizar FPS) is a representation of an informal proof in the formal Mizar language. A text in the Mizar FPS is between a fully checkable

[22] proofs by definition expansion, conditional proof, proofs by "reduction ad absurdum", proofs "per cases", proofs of existential statements

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.

The formal language of the Mizar FPS is a simplified version of the original Mizar Language. Moreover it concentrates mainly on theorems and proofs. Its formal representation is shown below, and is taken from the original work of F. Wiedijk [Wie04a]:

"if we specialize the notion of a formal proof sketch to the Mizar proof language, we have the following *formal proof sketch grammar*":

statement	=	proposition justification
	|	[*label* **:**] *term* = *term* justification
		{**.=** *term* justification}
proposition	=	[*label* **:**] formula
formula	=	*formula*
	|	**thesis**
justification	=	[**by** *label* {**,** *label*}]
	|	**proof** {step **;**} [cases] **end**
step	=	[**then**] statement
	|	**assume** proposition
	|	**let** *variable* {**,** *variable*}
	|	(**thus** | **hence**) statement
	|	[**then**] **consider** *variable* {**,** *variable*}
		such that proposition justification
	|	**take** *term* {**,** *term*}
	|	**set** *variable* = *term*
cases	=	**per cases** justification **;**
		{**suppose** proposition **;** {step **;**}}

The following listing is an example of Mizar FPS and it is an original work of F. Wiedijk presented in [Wie04a], and represents a proof of the Pythagoras' theorem shown in Figure 2.2:

```
theorem Th43: sqrt 2 is irrational
proof
 assume sqrt 2 is rational;
 consider a,b such that
4_3_1: a^2 = 2*b^2 and
  a,b are_relative_prime;                                    ←1
 a^2 is even;                                                ←2
 a is even;                                                  ←3
 consider c such that a = 2*c;                               ←4
 4*c^2 = 2*b^2;                                              ←5
 2*c^2 = b^2;                                                ←6
 b is even;                                                  ←7
 thus contradiction;                                         ←8
end;
```

3.9 Conclusion

In this chapter we gave a short overview of the Mizar system, which is a *proof system* designed for computer-aided formalisation of mathematics. In the first Section 3.1 of this chapter we presented an overview of the whole Mizar project, which lead to the development of different aspects of the Mizar. We briefly presented the Mizar Language in Section 3.2, which mathematicians use to formalise the knowledge of their papers. Furthermore in Section 3.3 we described the Mizar document, called *Mizar article* by the Mizar community, which is written using the Mizar language. We discussed the Mizar system in Section 3.4, which is the only implementation of the software for verifying texts written in the Mizar Language. We also illustrated the process of checking a Mizar article and covered that illustration with a number of examples annotated in the Mizar Language.

Furthermore, we discussed the Mizar library in Section 3.5, which is distributed with the Mizar system. Moreover, we presented briefly Mizar types and constructions, as well as the Mizar semantic search engine – the MML Query. In Section 3.6 we also discussed formalisation attempts done by K. Retel, which resulted in a number of articles submitted to the MML and later on automatically translated into the English language and published in the "Formalized Mathematics journal".

In Section 3.7 we highlighted the use of Mizar system in a number of teaching experiments conducted at many universities. In some of these experiments, K. Retel has been involved heavily and placed important role on the tutor and teacher of the Mizar system. Finally in Section 3.8, we illustrated the Mizar Formal Proof Sketch, which will be used as one of the levels in the path of gradual formalisation from CML, through MathLang into Mizar. This chapter contains number of short examples taken directly or indirectly from the MML.

Chapter 4

Document Rhetorical aspect Design

In this chapter we define MathLang's Document Rhetorical aspect (DRa). A brief explanation of DRa was presented in Section 2.5. Section 4.1 presents a brief overview of the aspect and its goals and expresses the need of its existence. In the same section we also present existing tools that allow capturing the narrative structure of mathematical documents.
In Section 4.2 we describe the term *ontology* and define the DRa *ontology*. Section 4.2 provides a short explanation of the annotation process required by the author wanted to annotate the DRa aspect. We present the DRa abstract and concrete syntax in Section 4.4. We illustrate in Section 4.5 the *dependency graph* which is achieved from the annotated document. We present the automatic transformation of the annotated document into a more pleasant visual display of the annotation in the form of the *dependency graph*. The following Section 4.6 defines the *logical precedence* and the *graph of logical precedences*, which is automatically generated from the *dependency graph* of the annotated document. Finally in Section 4.7 we provide a set of rules for analysing the well-formedness of DRa annotations through the verification of the *dependency graph* and the *graph of logical precedences*.

DRa is a MathLang aspect oriented toward capturing the narrative structure of mathematical documents. The DRa aspect design and development is the main contribution of this thesis. Due to the development of the path toward the Mizar proof assistant, the DRa becomes the main area of investigation within this PhD course. It is also important to remark, that the DRa development allows us to clarify slightly the path toward other *proof systems*.

4.1 Overview

There are many styles for capturing the narrative structure of a mathematical document. Each mathematician has its own conventions and traditions about labelling portions of texts (e.g., *chapter*, *section*, *theorem* or *proof*) and identifying statements according to their logical importance (e.g., one can say that *theorem* is more important than *lemma*). Furthermore, such narrative/structuring labels guide the reader's navigation of the text and form the key components in the reasoning structure of the theory reflected in the text. Moreover there is also a very important and clear approach of expressing the relations between mathematical components. This shows the logical dependencies within the document which help the reader recognize the theory structure of a paper before reading the details.

The reader could find his way while reading the document depending on how the structure and dependencies are expressed. One could produce a clear structure of a document by specifying explicitly where the important parts (e.g., sections, definitions, etc.) start and end, and also by expressing clearly what are the dependencies within the document. In such case, the reader has a clear view of the theory in the document. For example the left hand side of Figure 4.1 presents a clear and concise style of writing mathematics.

Some styles of writing mathematical texts resemble a newspaper-like writings. In such case, the reader usually have difficulties finding his way in the document. In the example presented on Figure 4.2, W. Sierpiński [1] uses a different font style (i.e., *italic*) to distinguish important parts of the theory presented in the document. In addition, the author does not mark explicitly the boundaries of the proofs. This requires from the reader a much careful study of the document in order to find those boundaries.

Another example, presented in Figure 4.3, by the same author W. Sierpiński, introduces a different way of expressing the narrative structure of mathematical document. In this example the author use clear labels and annotates clear boundaries of chunks of text. W. Sierpiński presents a number of exercises annotated in one section labelled "*exercises*". Each exercise expresses a problem that has to be solved. Below each exercise, the author provides a solution of that problem and annotates it as *proof*. Moreover, the author sometimes makes a *remark* to some *proofs*. This style of writing is different than the one we are used to, where a section regarding exercises merely lists mathematical problems, sometimes followed by a

[1] Note that W. Sierpiński was known for outstanding contributions to set theory. He was very productive and published over 700 papers and 50 books.

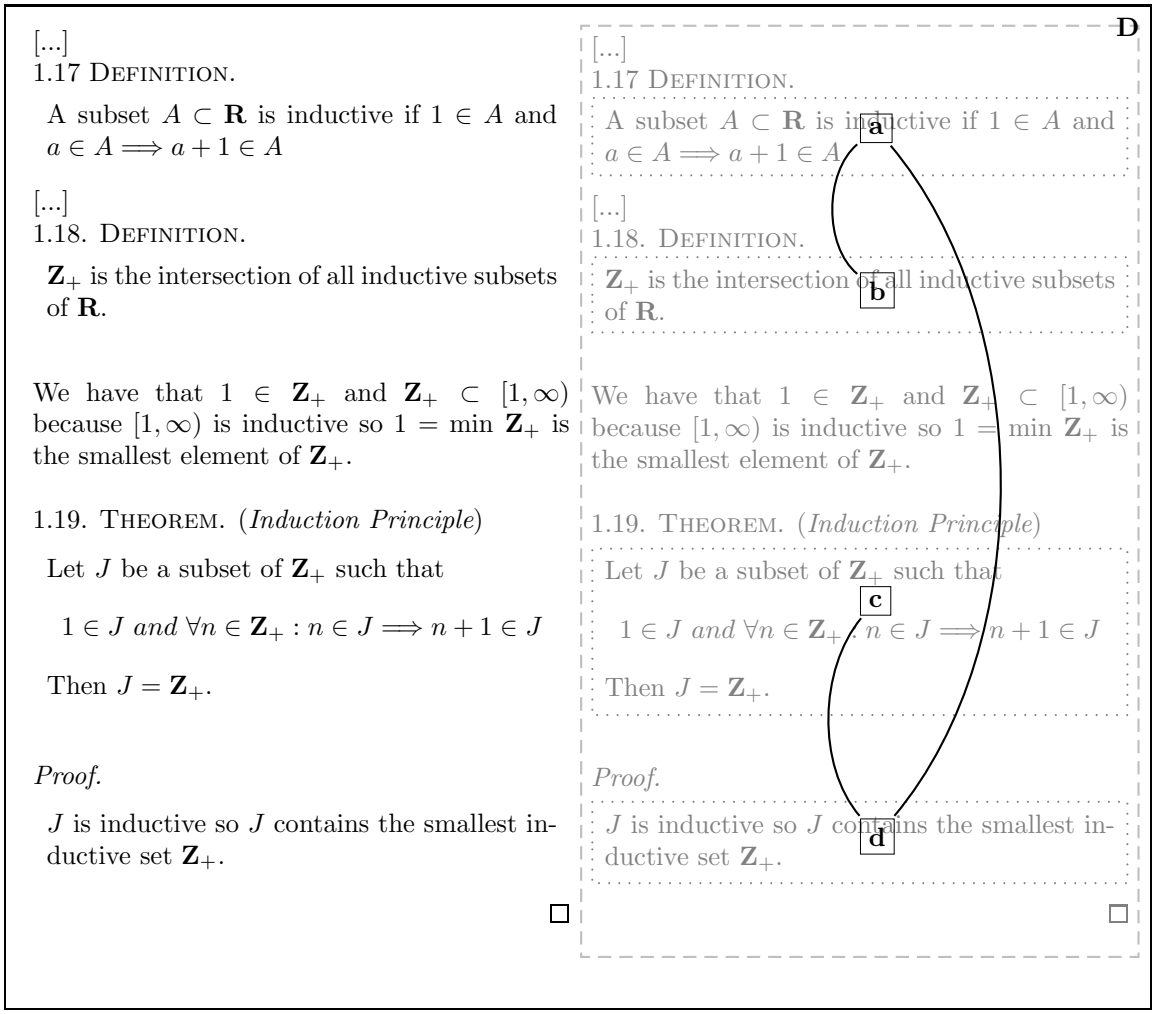


Figure 4.1: A fragment of the CML text with and without annotated DRa boxes around chunks of text and edges between these boxes.
The original text [Mol07, Chapter III, §2] of the given example is taken from J.M. Möller's notes [Mol07] regarding general topology and is reproduced on the left hand side of the figure. 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 or structural rhetorical roles.

[...]

We prove that *two congruences can be added or subtracted from each other provided both have the same modulus.*

Let

$$a \equiv b \pmod{m} \quad \text{and} \quad c \equiv d \pmod{m}. \tag{4.1}$$

In order to prove that $a+c \equiv b+d \pmod{m}$ and $a-c \equiv b-d \pmod{m}$ it is sufficient to apply the identities

$$a + c - (b + d) = (a - b) + (c - d) \text{ and } (a - c) - (b - d) = (a - b) - (c - d).$$

Similarly, using the identity

$$ac - bd = (a - b)c + (c - d)b,$$

we prove that congruences (2) imply the congruence

$$ac \equiv bd \pmod{m}.$$

Consequently, we see that *two congruences having the same modulus can be multiplied by each other.*

[...]

Figure 4.2: A short CML example from Congruence Theory by W. Sierpiński in [Sie64, Chapter V, §1]

number of hints but rarely provides full solutions to these problems.

Different styles of writing mathematics do not support a single way of expressing the narrative structure of mathematical documents. Each author has his own way of writing mathematics. There is an ongoing challenge to finding a standard way of expressing the narrative structure of mathematical documents. There exists a number of different tools that support and provide a standard way of expressing such structure. However, our approach is different as is explained further in this chapter. In the first three sections we discuss three different tools.

4.1.1 The DocBook format

DocBook[2] is a markup language and standard for publishing structured documents. According to [WM99], "it is particularly well-suited to books and papers about computer hardware and software, though it is by no means limited to them." DocBook provides a number of semantic element tags, that might be divided into a number of categories:

- sets - a collection of one or more books,

[2] http://www.docbook.org

EXERCISES. **1.** Prove that every natural number > 11 is the sum of two composite numbers.

Proof. Let n be a natural number greater than 11. [...]

2. Prove that there exist infinitely many natural odd numbers which cannot be represented as the sum of less than three numbers.

Proof. Such are, for instance, the numbers $(14k + 3)^2$, where $k = 1, 2, \ldots$ In fact, the numbers themselves are not primes. They cannot be represented as the sum of two primes either; for, if they could, then since they are odd, one of the primes would be equal 2, which would give $(14k + 3)^2 = 2 + p$, where p would be prime. Hence $p = 7(28k^2 + 12k + 1)$, which is impossible.

Remark. It can be proved elementarily that there exist infinitely many odd numbers which are sums of three different primes but are not sums of less than three different primes (cf. Sierpiński !!citation).

3. [...]

Figure 4.3: An example of some prime numbers problems presented by W. Sierpiński in [Sie64, Chapter III, §4]

- books - a collection of dedication (e.g., page occurred at the beginning of a book), navigation components (e.g., Table of Content, Index, List of Figures, etc.), divisions, components, etc.,
- divisions - a collection of parts and references,
- components - a collection of chapter-like elements of a book,
- sections - a collection of block elements and/or sections,
- meta-information - all of the elements at the section and higher levels, including a wrapper for meta-information about the content, e.g., author, title, publisher etc.,
- block elements - a paragraph-level element, e.g., paragraphs, examples, figures, etc.,
- inline elements - a collection of elements that are used to mark up some pieces of text; they are used to change the font size or style, and to make other small changes. However they do not cause any line or paragraph breaks.

From the point of view of capturing the narrative structure of the document we concentrate on the above categories except a the inline elements. Let us discuss in more detail the sectioning elements. The DocBook standard conforms to a number of different ways of sectioning passages of texts. To annotate a section, the author can use numbered element tags as: `<Sect1>`,...`<Sect5>` or un-numbered ones, like `<Section>` or `<SimpleSect>`. The first type of sections (i.e., `<Sect1>`,...,`<Sect5>`) introduces five levels of sectioning. Each of the section has to be properly nested, that is `<Sect2>` has to be contained in `<Sect1>`, `<Sect3>` in `<Sect2>`, etc. Ob-

viously this causes limitation to the depth of sectioning. Therefore, the second way of sectioning, an alternative to numbered sections, was introduced, using a `<Section>` element tag. The `<Section>` element is recursive, meaning that you can nest it to any depth desired. The last way to introduce sections is by using the `<SimpleSect>` element. As the name suggests, it is a simple section that can occur at any level, but cannot have any other sectioning elements nested within it. For illustration purposes of DocBook we present a small example in Listing 4.1.

```
<!DOCTYPE chapter PUBLIC "-//OASIS//DTD DocBook V3.1//EN">
<book>
  <title>Example book
  <chapter label="6" id="test-chapter">
    <title>Numbered sections
    <para>Presents an example of usage of numbered sections
    <sect1>
      <title>Top Level Section - Sect1
      <para>First paragraph in Sec1
      <sect2>
        <title>title of Sect2
        <para>First paragraph in Sec2
  <chapter>
    <title>Test Chapter
    <para>This chapter uses recursive sections.
    <section>
      <sectioninfo>
        <abstract>
          <para>A trivial example of recursive sections.
      <title>Like a Sect1
      <subtitle>Learning recursion...
      <para>paragraph text
      <section>
        <title>Like a Sect2
        <para>This section is like a Sect2.
        <section>
          <title>Like a Sect3
          <para>This is another recursion
```

Listing 4.1: A DocBook annotation example.

For readability and brevity, we show only the opening tag of each XML element; we use indentation to express nesting.

The DocBook division elements are self-descriptive and some of the names of tags contain a hierarchical level (depth) of the elements. It does not differ a lot from

the scientific LaTeX format for presenting the narrative structure of a document. In LaTeX, each division element has its own name (e.g., part, chapter, section, subsection etc.), moreover the TeX programs allow to analyse if the annotated parts of the text are well embedded, for example if 'section' is used inside a 'chapter' and not inside a 'subsection'. There are similar validations that we can achieve with the DocBook annotation, by using XML tools and the DocBook DTD or Schema.

4.1.2 The Text Encoding Initiative Guidelines

Another standard for representing texts in digital form has been developed. The Text Encoding Initiative is a consortium which develops a set of Guidelines which specifies encoding methods for machine-readable texts. Similarly to DocBook, it was first rooted in SGML[3] and at present developed in XML format. It is mainly oriented toward annotating and encoding literary documents in humanities, social sciences and linguistics. According to the TEI website[4], "the TEI Guidelines have been widely used by libraries, museums, publishers, and individual scholars to present texts for online research, teaching, and preservation."

The TEI differs from DocBook and takes another approach for annotating documents. One reason is that DocBook is specifically designed for computer hardware and manuals, whereas TEI concentrates on literature. The TEI Guidelines are expressed as modular and define a number of modules, each of which declares particular XML elements and attributes. Using this modular approach one could customise and construct its own TEI schema using any combination of modules. However, some TEI modules are core and are mandatory to all customised schema. A main example of the TEI customised schema is TEI Lite[5]. Any TEI customised schema conforms to TEI element tags which might be used to annotate the document. Such annotated document can be later on validated, similarly to DocBook, using XML tools.

Each document annotated using TEI schema is either a single document, where the first element tag is `<TEI>`, or a collection of documents, where the first element tag is `<teiCorpus>`.

```
<teiCorpus>
  <teiHeader>...
  <TEI>
    <teiHeadr>...
```

[3]SGML stands for Standard Generalized Markup Language
[4]http://www.tei-c.org/index.xml
[5]http://www.tei-c.org/Guidelines/Customization/Lite/

```
    <text>...
  <teiCorpus>
    <teiHeader>...
    <TEI>
      <teiHeader>...
      <text>...
    <TEI>
      <teiHeader>...
      <text>...
```

Listing 4.2: The TEI example presenting the main document or a collection of documents.
For readability and brevity, we show only the opening tag of each XML element; instead we use indentation to express nesting.

The `<teiHeader>` is a mandatory tag which supplies the descriptive and declarative information about the text itself, its source, its encoding, and its revisions. It also provides an electronic analogue to the title page attached to a printed work.

The `<text>` element tag, contains a single text of any kind, whether unitary or composite, for example a poem, a collection of essays, etc. The default overall structure of any `<text>` is defined by the following elements, as discussed on the TEI website subpage [6]:

- `<front>` - (front matter) contains any page found at the start of a document, before the main body, e.g., title page, dedication, preface, etc.,
- `<body>` - contains the whole body of a text without its front and back matter,
- `<group>` - groups together a sequence of distinct texts (or groups of such texts) which are single unit texts, e.g., the collected works of an author, a sequence of novels, etc.,
- `<back>` - (back matter) contains any appendixes.

The `<body>` of a document can be divided into a number of chunks of text, which form a hierarchical textual divisions and subdivisions, such as chapters or sections. As mentioned above, these divisions and subdivisions vary depending on the style of the author writing the document. For instance a major subdivision of a book will be 'chapter', of a report is usually called 'part' or 'section', etc. Similarly, texts which are not organised as linear prose narratives, or not as narratives at all, will frequently be subdivided in a similar way: a drama into 'acts' and 'scenes', a diary or a day book into 'entries', a newspaper into 'issues' etc.

Because of this variety, the TEI Guidelines propose that all textual divisions

[6] http://www.tei-c.org/release/doc/tei-p5-doc/en/html/DS.html

will be encoded using the same named element tag with an attribute type used to provide the hierarchical level of such annotated element. Similarly to the DocBook sectioning element, the TEI provides numbered (i.e., `<div1>`,...,`<div7>`) and unnumbered (i.e., `<div>`) division element tags. Apart from the division element tag, the TEI introduced another tag for annotating paragraphs, i.e., a tag named `<p>`. All of this group of elements uses three types:

1. `type` - which indicates the conventional name for a category of this element; it also indicates the hierarchical level of the element, e.g., 'book', 'part', 'chapter', 'section',
2. `xml:id` - which specifies a unique identifier of that element within the whole document,
3. `n` - which specifies a short name or a number for the division.

For illustration purpose of the usage of TEI Guidelines, we present a short example in Listing 4.3.

```
<TEI>
  <teiHeader>...
  <div1 type="book" n="I" xml:id="L010000">
    <head>Book I
    <div2 type="chapter" n="1" xml:id="L010100">
      <head>Of writing lives in general,...
      <p>This chapter describes...
    <div2 type="chapter" n="2" xml:id="L010200">
      <head>DRa description, ...
      <p>This chapter describes ...
      <div3 type="section" n="2.1" xml:id="L010201">
        <p>section...
    <trailer>The end of the first Book...
  <div type="book" n="II">
    <head>Book II
    <div type="chapter" n="1">
      <head>Of divisions in authors
      <p>...
    <div type="chapter" n="2">
      <head>...
      <p>...
      <div type="section" n="2.1">
        <p>...
```

Listing 4.3: The example presenting the usage of TEI Guidelines.

For readability and brevity, we show only the opening tag of each XML element; instead

we use indentation to express nesting.

4.1.3 The OMDoc format

Similarly to DocBook and TEI Guidelines, where each of them is oriented toward different audiences or different areas of science, there was a need to develop a standard format for representing mathematical knowledge. The OMDoc [7], created by M. Kohlhase, is a semantic markup format and data model for Open Mathematical Documents , see [Koh06b]. It was developed to cover the context and content of a whole range of mathematical documents using a standard way. The OMDoc format is aimed to be a communication medium between the presentation layer of the mathematical knowledge on the one hand and the integration of such knowledge among external mathematical reasoning systems. At present OMDoc is used in a number of different projects, for example in e-learning, in data exchange between various *theorem provers* and *computer algebra systemCAS*, and as a base format for later presentation-oriented format.

OMDoc presents mathematical knowledge on three levels [Koh06a]:

The Theory Level: At this level, OMDoc annotates a collection of statements into theories, and specifies relations between theories. Theories may import each other and therefore a former theory might be reusable in a later developed theory. OMDoc theories might be seen as OpenMath content dictionaries [BCC+04, Dav02].

The Statement Level: This level is oriented toward the structure of mathematical statements. It mainly focuses on making explicit the narrative structure of mathematical documents by expressing precisely statements like theorems, definitions, proofs, examples and relations among them (e.g., "this theorem is proved by this proof").

The Object and Formula Level: At this level OMDoc uses an OpenMath and Content-MathML. Both are well established standards which mainly focus on specifying the meaning of mathematical objects and formulas, and focus on a content and context markup for the structure of objects and formulas.

OMDoc uses different ways of annotating mathematical statements. The first approach provides a rough classification of mathematical statements, whereas the second approach is more oriented towards making explicit the contribution of mathematical statements among theories and the interaction of those statements within

[7] `http://www.omdoc.org`

mathematical contexts.

OMDoc provides two possible pproaches for annotating narrative information of a passage of text.

The first approach OMDoc uses the `omtext` element tag to mark up text fragments that form conceptual units, e.g., definitions, paragraphs, statements or remarks. This element tag has an attribute `type` which classifies the chunks of text by their rhetorical role. The `type` attribute can have one of the following values: 'abstract', 'introduction', 'conclusion', 'comment', 'axiom', 'definition', 'example', 'proof', 'derive' (a step in a proof), 'hypothesis' (local assumption in a proof), etc. Finally OMDoc also reserves values: 'theorem', 'lemma', 'corollary', etc.
By the usage of the `omtext`, the author provides a rough classification of mathematical statements within the theory presented. For an explicit marking of mathematical statements, that contributes to the theory level and interacts with mathematical contexts, OMDoc uses different and more specific tag elements.

The second approach, OMDoc uses more specific element tags for annotating mathematical statements. The OMDoc markup distinguishes the knowledge elements of a theory into *constitutive* ones like symbols, axioms, and definitions (which present the essence of the annotated theory) and *non-constitutive* ones such as assertions, their proofs, examples (which illustrate properties and attributes of mathematical objects determined by the constitutive statements).

The *constitutive* statements of a theory can be annotated in OMDoc using the following element tags: `axiom`, `definition`, `symbol` and `type`. Names of those element tags are self-descriptive, therefore we do not explain them in here. The remaining statements play the role of *non-constitutive* ones on the theory level. They are marked using the following element tags: `example`, `alternative` and `assertion`. The `example` is used to mark mathematical examples within the theory, the `alternative` is used to mark the alternative statement to the one that is being annotated, for instance we can provide a definition of a symbol and later on provide an alternative definition of the same symbol. Finally, and most interestingly, the `assertion` tag is used to mark any mathematical statement. The `assertion` element tag contains the attribute `type` which classifies the mathematical statement. This `type` attribute can use the following values: 'theorem', 'proposition', 'lemma', 'corollary', 'postulate', 'conjecture', 'false-conjecture', 'obligation', 'assumption' and 'formula'.

As an illustration of an OMDoc document we present the following example:

```
<omdoc xmlns="http://www.mathweb.org/omdoc"
```

```
  xmlns:dc="http://purl.org/dc/elements/1.1/"
  version="1.2">

  <theory xml:id="test-theory-id">
    <omtext xml:id="intro" type="introduction">
      This is a test example
    </omtext>

    <symbol name="one">...</symbol>
    <definition xml:id="one.def" for="#one" ...>
      <CMP>0 is natural number ... </CMP>
    <FMP>...</FMP>
    </definition>

    <assertion xml:id="a1" type="theorem">
      <CFP>For all x being natural number x != S(x).</CFP>
      <FMP>....</FMP>
    </assertion>
    <proof for="#a1">...</proof>

    <example xml:id="mon.ex1" for="#monoid" type="for"
             assertion="string.struct.monoid">...</example>
  </theory>
</omdoc>
```

Listing 4.4: An example which presents the usage of OMDoc standard. OMDoc *provides additional element tags:* `CFP` *(i.e., commented mathematical property) and* `FMP` *(i.e., formal mathematical property). The former one is used for expressing the informal representation of the formula. The latter one is used for representing the formal mathematical content in the form of* OpenMath *objects.*

4.1.4 Why do we need DRa?

All the above described formats provide a standard way of annotating documents using XML techniques. Their usage has been developed to suit various types of documents from literature, poetry, computer manuals and hardware documentation, finally to mathematical papers, books, tutorials etc. Those standards are used for representing documents in the machine-readable format, which can be later on processed to achieve any desired level of presentation. This is available due to the XML representation of documents and the XSL transformation tools power.

All these systems allow to separate/divide a document into a number of struc-

tural components (sections or mathematical assertions) which can be annotated in the computerised version. However, we have been developing our own standard. Our proposed markup system is simpler and is concentrated only on the annotation of the narrative structure of mathematical documents, whereas others are more oriented towards capturing most of the subtleties of documents.
We believe that there are some limitations to the above formats and moreover, using those formats is still labour-intensive due to the complexity of the XML schema provided by each system, and due to a number of possible ways of annotating documents (by using different element tags).

Our own developed format provides an easy and flexible way for annotating the rhetorical structure of mathematical documents. For example it allows to annotate a group of statements according to their narrative structural role played within the document and at the same time it provides a mathematical label. For instance one could annotate a big chunk of text within a chapter and assign a division element name called 'section', and at the same time it could express the mathematical role of such section to be a 'theorem'.
At present none of the above standards provide a nice way of annotating such chunk of text without the need to use two different tags annotated on top of each other, i.e., `<section><assertion type="theorem">....` This is one of the limitation and complexity of the above representation formats of documents.

Moreover, the MathLang framework provides a flexible way of annotating different aspects independently at any order or simultaneously during one annotation process. This provides extra flexibility with enhancement of document annotation and computerisation. Again, above systems does not provide such support.

4.2 The Annotation System Ontology

Looking at different styles of mathematical knowledge representation we can distinguish two kinds of document structural units: *division elements* and *mathematical units* ([KMRW07b]). *Division elements* express a textual structure (e.g., chapter or section) of mathematical texts. Whereas, *mathematical units*, are usually expressed in mathematical textbooks and papers in terms of theorem, lemma or remark. Some *mathematical units*, for instance "proof", are more or less hinted by the authors' style of writing (see for example Figure 4.2). The human reader is able to recognise and infer them only by looking carefully at the original text.

We express and tag these structural units, *division elements* and *mathematical*

units, explicitly. By making explicit annotations of structure units we refine the content of the already captured original text, and at the same time we give a wider possibility for (semi)automatic text manipulation (see Sections 4.5 and 5.3.2).

To express the DRa system we enhance our development by Semantic Web Technologies, such as Resource Description Framework (RDF) [LS99] or Web Ontology Language (OWL) [MvH04].

4.2.1 Ontology

The literature contains many definitions of an ontology. Roughly speaking, "an ontology is a specification of conceptualisation" [Gru]. An ontology is a representation of terms with their relationships in a specific domain. An ontology describes:

1. individuals/instances of a class: the basic objects, for example "Bach" is an instance of class "Person" [8],
2. classes/abstract groups: sets, or collections of objects, for example "Person",
3. relations/properties between objects, for example the relation *childOf* [9].

For example, "Sebastian Bach" could be linked to another person, his father, "Ambrosius Bach" by the relation typed "is a child of". In terms of RDF, this can be expressed by the following "subject-predicate-object" triple:

("Sebastian Bach", *isChildOf*, "Ambrosius Bach").

Throughout the rest of the chapter we will use the RDF triples to express DRa annotated relations. We use them in the above presented format: (*subject*, *predicate/relation*, *object*).

4.2.2 DRa ontology in a nutshell

To model our DRa ontology we used the OWL-DL Web Ontology Language, which is the OWL sub-language so-named due to its correspondence with *description logics* [MvH04]. An OWL ontology may include a description of classes, instances of them and properties between their elements.
The information presentation using OWL is very powerful in the way that it is suitable for exchanging information and processing by other software applications.

[8] "The Friend of a Friend" (FOAF), as described on `http://www.foaf-project.org/`, is "creating a Web of machine-readable pages describing people, the links between them and the things they create and do."

[9] `http://vocab.org/relationship/` – A vocabulary for describing relationships between people

Following OWL, our DRa ontology makes explicit in a domain of the DRa, the formal description of:

- classes – whose names start with a capital letter, e.g., StructuredUnit,
- individuals – which are elements of classes, e.g., section,
- properties/relations – whose names start with a small letter, e.g., justifies or hasMathematicalRhetoricalRole.

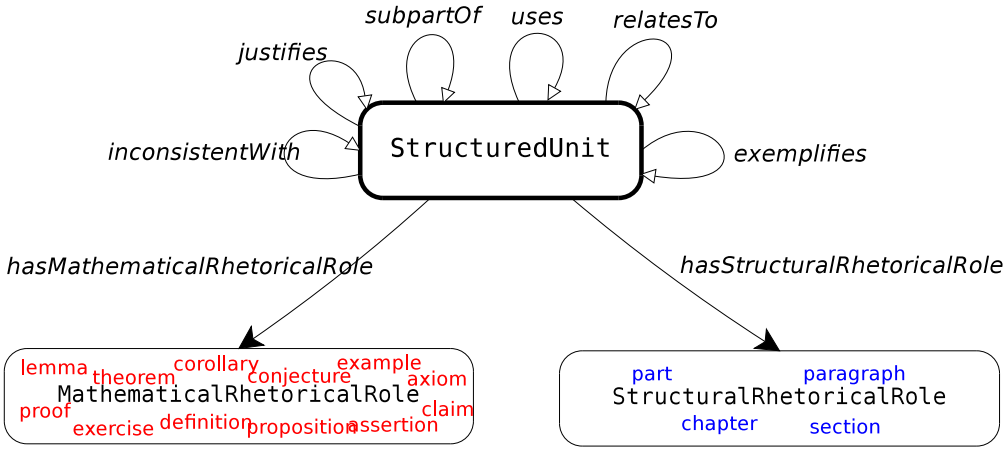


Figure 4.4: Part of the DRa annotation system ontology.

The DRa concepts are given as three OWL *classes* [MKSM04] (see Figure 4.4):

1. StructuredUnit – whose instances are types of any of the two classes below,
2. MathematicalRhetoricalRole – whose instances are lemma, proof, etc.
3. StructuralRhetoricalRole – whose instances are chapter, section, etc.

The above two classes MathematicalRhetoricalRole and StructuralRhetoricalRole are disjoint, and are subparts of the main class StructuredUnit. We describe them in more details in this section.

Relations between various instances of the above classes are given as OWL *object properties* [MKSM04]:

1. The ownership relation between structural units and the roles played in a text, i.e. hasMathematicalRhetoricalRole and hasStructuralRhetoricalRole.
   E.g., in Figure A.1, $(D1, \text{hasMathematicalRhetoricalRole}, \text{definition})$.

2. The relations between instances of the class StructuredUnit:

   (a) relatesTo, justifies, subpartOf, uses, exemplifies, inconsistentWith.

The relations of the first kind (item 1) are modeled as *object properties* (i.e., link individuals of one class to individuals of another class). The relations presented in

item (2) are modeled as *subproperties* of the generic *object property* – specifies, i.e. $(A, \text{specifies}, B)$, where A, B are instances of the class StructuredUnit.

Relations between instances of the classes MathematicalRhetoricalRole or StructuralRhetoricalRole and the XML schema datatype (xsd:string) are given as OWL *datatype properties* [MKSM04] (i.e., they link individuals of a class to the XML Schema datatypes [BM01]): hasOtherMathematicalRhetoricalRole and hasOtherStructuralRhetoricalRole. The existence of these relations gives the freedom to provide a new label not appearing in Table 4.5. This is possible through the usage of a variant property called hasOtherStructuralRhetoricalRole for *division elements* and hasOtherMathematicalRhetoricalRole for *mathematical units*. The range of values of such properties is restricted to the XML Schema datatype "string", so for example we can annotate the text using the following RDF triple
$(A, \text{hasOtherMathematicalRhetoricalRole}, problem)$.

4.2.2.1 The Instances of DRa Classes

Since both *division elements* and *mathematical units* express the boundaries of chunks of text, we included them into one class (StructuredUnit). The two disjoint classes: StructuralRhetoricalRole and MathematicalRhetoricalRole allow to represent the different roles played by *division elements* and *mathematical units*.

Instances of the first class, StructuralRhetoricalRole, are conventional names for *division elements* which might at the same time express the hierarchical level of a document structure, i.e., chapter, section, etc.
Instances of the class MathematicalRhetoricalRole are common labels and names for the *mathematical units*, i.e., theorem, corollary, etc. All instances of the classes StructuralRhetoricalRole and MathematicalRhetoricalRole, are fixed conventional labels used to annotate mathematical documents.

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, relatesTo

Figure 4.5: DRa annotations.

4.2.2.2 The DRa Relationships

The DRa ontology allows to relate a particular instance of the class StructuredUnit with any instance of StructuralRhetoricalRole and MathematicalRhetoricalRole via the properties hasStructuralRhetoricalRole and hasMathematicalRhetoricalRole respectively. We allow the use of both properties when relating to an instance of a class StructuredUnit. This enables to specify, for instance, that a passage of text plays the structural role "section" and concurrently plays the mathematical role "theorem". By stating two properties simultaneously in a document annotation we allow to encode different styles of writing mathematics.

While annotating the narrative feature of a document, we make explicit correlations between recognised chunks of text. For this, within the DRa ontology, we introduced other properties which describe relations between instances of the class StructuredUnit and represent dependencies between *mathematical units* and/or *division elements*. Our DRa ontology clarifies important relationships in a text. The properties used to represent relations between chunks of text, have human readable names: relatesTo, justifies, subpartOf, uses, inconsistentWith, exemplifies. In a formal system, some of these properties have formal meanings:

1. $(n_1, \mathsf{justifies}, n_2)$ – n_1 describes a proof object that proves the formula n_2.

2. $(n_1, \mathsf{uses}, n_2)$ – (1) All/some variables under the general quantifiers that have been applied in a formula n_2, have been instantiated in formula n_1 which could be proved via simple reasoning where n_2 appears among references needed to prove n_1. (2) The formula n_2 has been unfolded or folded in the formula n_1.

3. $(n_1, \mathsf{subpartOf}, n_2)$ – (1) if n_2 is a formula, then n_1 is an inseparable part of that formula; (2) if n_2 is a proof object, then n_1 is part of that proof object.

4. $(n_1, \mathsf{inconsistentWith}, n_2)$ – if n_1 and n_2 are proof objects of one formula, then the environment in which these proof objects were achieved is inconsistent.

4.3 The Annotation Process

This section expresses the annotation process that a mathematician has to follow to computerise his document with the MathLang DRa aspect. The annotation can be accomplished independently from the CGa and TSa aspects. It could be

performed either on the already annotated text with the CGa or TSa aspects, or could be performed on the original document.

We present the DRa aspect annotation process on the approach of H. Barendregt to the proof of the Pythagoras' theorem, as seen in Figure 4.6. This version of the proof is said to be more formal and concrete compared to the original version of the proof by G.H. Hardy and E.M. Wright [HW80].

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

Proof. Define on $\mathbb{N}$ the predicate:

$$P(m) \iff \exists n.m^2 = 2n^2 \ \& \ m > 0.$$

Claim. $P(m) \Longrightarrow \exists m' < m.P(m')$. Indeed suppose $m^2 = 2n^2$ and $m > 0$. It follows that m^2 is even, but then m must be even, as odds square to odds. So $m = 2k$ and we have

$$2n^2 = m^2 = 4k^2 \Longrightarrow n^2 = 2k^2$$

Since $m > 0$, if follows that $m^2 > 0, n^2 > 0$ and $n > 0$. Therefore $P(n)$. Moreover, $m^2 = n^2 + n^2 > n^2$, so $m^2 > n^2$ and hence $m > n$. So we can take $m' = n$.

By the claim $\forall m \in \mathbb{N}.\neg P(m)$, since there are no infinite descending sequences of natural numbers.

Now suppose $m^2 = 2n^2$ with $m \neq 0$. Then $m > 0$ and hence $P(m)$. Contradiction. Therefore $m = 0$. But then also $n = 0$. □

Corollary 1. $\sqrt{2} \notin \mathbb{Q}$

Proof. Suppose $\sqrt{2} \in \mathbb{Q}$, i.e. $\sqrt{2} = p/q$ with $p \in \mathbb{Z}, q \in \mathbb{Z} - \{0\}$. Then $\sqrt{2} = m/n$ with $m = |p|, n = |q| \neq 0$. It follows that $m^2 = 2n^2$. But then $n = 0$ by the lemma. Contradiction shows that $\sqrt{2} \notin \mathbb{Q}$. □

Figure 4.6: The proof of Pythagoras' theorem, of the irrationality of $\sqrt{2}$ – H. Barendregt's version presented in [Bar06]

The MathLang user who wants to annotate a document with all three MathLang aspects (i.e., CGa, TSa and DRa) uses the TEXMACS editor with the MathLang plugin. The annotation process does not require a lot of knowledge. Moreover, the original document can be annotated independently for all three aspects.

4.3.1 What does the user have to do?

To annotate a mathematical text, the user follows three easy steps:

1. He wraps chunks of text with boxes. Then he uniquely names each box. Unicity allows avoiding problems when stating relations between some boxes. If the user

Lemma 1.

For $m, n \in \mathbb{N}$ one has: $m^2 = 2n^2 \Longrightarrow m = \boxed{\mathbf{A}} = 0$

Proof.

Define on $\mathbb{N}$ the predicate: $P(m) \Leftarrow \boxed{\mathbf{E}} \exists n.m^2 = 2n^2 \ \& \ m > 0$.

Claim. $P(m) \Longrightarrow \exists m' \boxed{\mathbf{F}} < m.P(m')$.

Indeed suppose $m^2 = 2n^2$ and $m > 0$. It follows that m^2 is even, but then m must be even, as odds square to odds. So $m = 2k$ $\boxed{\mathbf{G}}$ and we have $2n^2 = m^2 = 4k^2 \Longrightarrow n^2 = 2k^2$ Since $m > 0$, if follows that $m^2 > 0$, $n^2 > 0$ and $n > 0$. Therefore $P(n)$. Moreover, $m^2 = n^2 + n^2 > n^2$, so $m^2 > n^2$ and hence $m > n$. So we can take $m' = n$. $\boxed{\mathbf{B}}$

By the claim $\forall m \in \mathbb{N}.\neg P(m)$, since there are no infinite descending sequences of natural numbers.

Now suppose $m^2 = 2n^2$

with $m \neq 0$. Then $m > 0$ and hence $\boxed{\mathbf{H}}(m)$. Contradiction.

Therefore $m = 0$. But then also $n = \boxed{\mathbf{I}}$ □

Corollary 1. $\sqrt{2} \boxed{\mathbf{C}} \mathbb{Q}$

Proof. Suppose $\sqrt{2} \in \mathbb{Q}$, i.e. $\sqrt{2} = p/q$ $\boxed{\mathbf{D}}$ $p \in \mathbb{Z}, q \in \mathbb{Z} - \{0\}$. Then $\sqrt{2} = m/n$ with $m = |p|, n = |q| \neq 0$. It follows that $m^2 = 2n^2$. But then $n = 0$ by the lemma. Contradiction shows that $\sqrt{2} \notin \mathbb{Q}$. □

Figure 4.7: The presentation of Figure 4.6's example with annotated and uniquely named boxes, which contains hidden attributes, i.e., mathematical roles, assigned to each box name.

makes a mistake and annotates two boxes with the same name, the validation of the DRa aspect will provide a warning saying that it is not allowed. This will be discussed in more details in Section 4.4. For our example of Figure 4.6, the names of those boxes are: $A, B, C, D, E, F, G, H, I$. The view of our example with annotated and named boxes is shown on Figure 4.7.

2. He assigns to each (name of a) box, structural or/and mathematical rhetorical roles which this box may play. He can either use the structural/mathematical roles listed in Table 4.5, or specify his own. He uses the RDF triples approach to assign a role to a specific uniquely named box. For our example of Figure 4.7, we assigned the roles as stated in the left hand column of Table 4.1.
3. He makes explicit the relations between wrapped chunks of texts using the relation names of Table 4.5. For our example of Figure 4.6, the relations could be assigned as presented on the second column of Table 4.1. The relations are presented as visible and labelled arrows in Figure 4.8.

Assigned rhetorical roles	Relations
$(A, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{lemma})$	$(B, \mathsf{justifies}, A)$
$(B, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{proof})$	(F, uses, E)
$(C, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{corollary})$	(G, uses, E)
$(D, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{proof})$	$(G, \mathsf{justifies}, F)$
$(E, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{definition})$	(H, uses, E)
$(F, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{claim})$	$(H, \mathsf{subpartOf}, B)$
$(G, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{proof})$	$(I, \mathsf{subpartOf}, B)$
$(H, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{case})$	(D, uses, A)
$(I, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{case})$	$(D, \mathsf{justifies}, C)$

Table 4.1: Annotation of the example from Figure 4.6 presented as RDF triples.

We use RDF triples [LS99] to represent the relationships between the boxes annotated by the mathematician. Each triple is expressed by a *subject-predicate-object* triple, where a *predicate* (i.e., a property) denotes a relationship. The order in a triple between *subject* and *object* is significant, and when transformed into a *dependency graph* the direction of the arc the triple makes, always points toward the *object*.

As said earlier, the mathematician uses the TEXMACS editor with the MathLang plugin to annotate mathematical document with all three aspects of the framework. The view of the document computerised with the DRa aspect on top of the original document using the TEXMACS editor is different from the one presented above. The above example was used to present and provide an idea of the annotation process required by the mathematician to be performed for annotating the DRa aspect. The original view of the annotated document with the DRa nodes

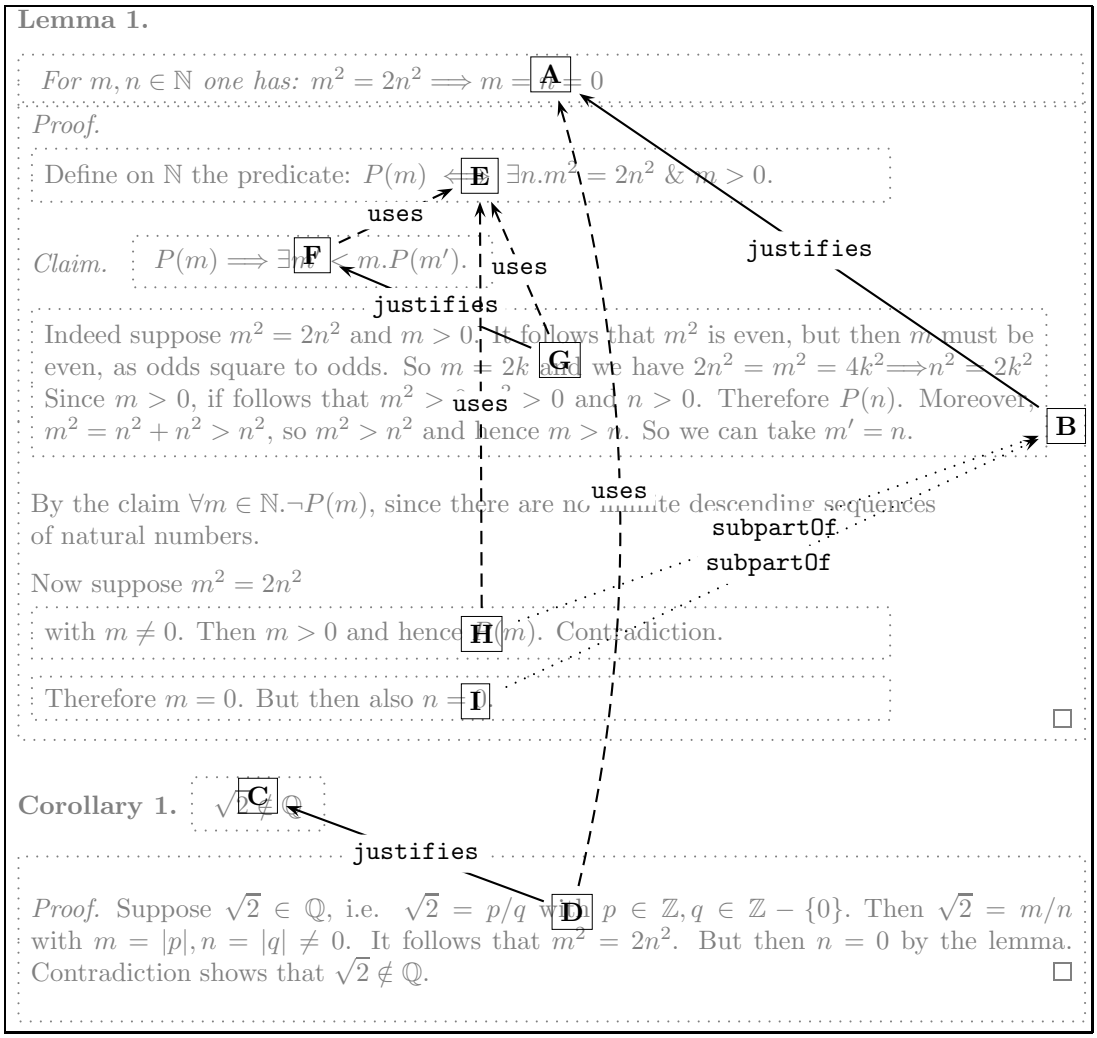


Figure 4.8: The presentation of Figure 4.6's example with uniquely named boxes, annotated and made visible relations on top of the original CML view of the example.

and relations is presented on Figure A.6.

4.4 Plain and Concrete Syntax

In this section we present the *plain* and *concrete syntax* of the MathLang DRa aspect. The *plain syntax* is closely related to the one that was described and presented through a number of examples for the MathLang CGa, see Section 2.3. Furthermore, we use the MathLang *plain syntax* as well as the *concrete syntax* (XML) and the TEXMACS preview across the whole thesis for presentation purposes. We use a number of toy examples to present the usage of both, *plain* and *concrete* syntaxes.

It is important to remind the reader that the annotation process of the DRa aspect could be independent from the annotation of other MathLang aspects. However, for clarity purposes we annotate DRa on top of the already computerised version of the document with the CGa aspect. Reason for that is, that we explain the *plain* syntax which is highly correlated with the CGa's *plain* syntax, and both looks in a way that the "human" can approach to read them.

Furthermore, the MathLang user should note that the *plain syntax* is not intended, at any point, to be read and used by the user to annotate the DRa aspect. However, it is used in this thesis for annotation presentation purposes as well as for providing a clear view of the examples. Furthermore, the MathLang checker[10], checks a document written in the *plain syntax* and at the same time transforms it to the XML format which is stored in the file. Similarly to the *plain syntax*, the MathLang document can be annotated using the XML schema, however it is very tedious and labour intensive. Therefore the user/mathematician who wants to annotate the document with the MathLang framework uses the TEXMACS plugin. We will present in more details the annotation of the DRa aspect using the MathLang plugin in Section 6.2.

4.4.1 Plain syntax

For the *plain syntax* we use similar notations to the one that were discussed and presented through examples for the MathLang CGa aspect annotation. We also use the `typewriter` font for printing DRa examples.

[10]The MathLang checker has been originally implemented in OCaml by M. Maarek and presented in M. Maarek PhD thesis. Another implementation of the MathLang checker has been developed by C. Zengler using modern JAVA programming language.

There exist two constructions for the DRa aspect annotation in the *plain syntax*. First it is used for wrapping boxes around chunks of text and to uniquely name them. The second one is used for assigning roles for previously annotated boxes as well as for stating relations between them.

The following representation of the DRa annotation is expressed in the abstract syntax format. This format is taken from the M. Maarek's PhD thesis [Maa07]. As described above, the DRa annotation is performed either on the CGa *step* or on the level of the MathLang document.

mathlang-document ::=	*step*	Mathlang document
step ::=	*phrase*	Basic step
|	*step* ▷ *step*	Local scoping
|	$\{\overrightarrow{step}\}$	Block
|	*dra-annotation*	DRa annotation
dra-annotation ::=		
	`[node did@`*node-id step* `]`	DRa annotated step
|	`[about did@`*node-id* `hasMRR@`*role*`]`	Mathematical role
|	`[about did@`*node-id* `hasSRR@`*role*`]`	Structural role
|	`[about did@`*node-id* `hasOMRR@`*role*`]`	Other Mathematical role
|	`[about did@`*node-id* `hasOSRR@`*role*`]`	Other Structural role
|	`[did@`*relid* `src@`*node-id* `type@`*reltype* `anc@`*node-id*`]`	DRa relation

4.4.1.1 Boxes annotation

The boxes presentation describes the way to annotate the CGa steps with a DRa box on top of the CGa step.

`[node did@`*node-id step* `]`

The following format consists of the indication of the DRa node (i.e., `node`), the unique id of the node (i.e., `did`) and finally a CGa *step*.

The example shown in the Listing below presents a possible annotation of the box around the "Corollary" from Figure 4.6.

```
[node did@C not( in(sqrt(2),Q) );];
```

Listing 4.5: The *plain syntax* annotation for the Corollary from Figure 4.6

Similarly, the "Lemma" from the same example and Figure 4.6 could be annotated with the DRa on top of the CGa, as follows:

```
[node did@A
 forall(m:N, forall(n:N,
  implies( =(sq(m),*(2,sq(n))), and(=(m,n), =(n,0)) ))); ];
```

Listing 4.6: The *plain syntax* annotation for the Lemma from Figure 4.6

4.4.1.2 Boxes description and Relationships annotation

For each annotated DRa node, as described above, we assign an attribute with its mathematical or/and structural role. To do so we follow the RDF approach with the *plain syntax* as presented in Figure 4.9.

1. `[about@`*node-id* `hasMRR@`*node-mathematical-role*`];`
2. `[about@`*node-id* `hasSRR@`*node-structural-role*`];`
3. `[about@`*node-id* `hasOMRR@`*node-other-mathematical-role*`];`
4. `[about@`*node-id* `hasOSRR@`*node-other-structural-role*`];`
5. `[did@`*relation-id* `src@`*source-node* `type@`*relation-type* `anc@`*anchor-node*`];`

Figure 4.9: The *plain syntax* annotation rules for the DRa nodes description and relationships.

The first four annotation rules (1–4, presented on Figure 4.9) are used to annotate the narrative role which the *node-id* plays in the original document. For instance, the DRa node annotated on Listing 4.5 and originally presented on Figure 4.6 could have been assigned the mathematical role "corollary" and could have been annotated as follows.

```
[about@A hasMRR@corollary];
```

Listing 4.7: The DRa "corollary" node attribute assignment using *plain syntax*.

Similarly, the "lemma" DRa node from Listing 4.6 could be annotated as follows.

```
[about@A hasMRR@lemma];
```

Listing 4.8: The DRa "lemma" node attribute assignment using *plain syntax*.

In Listing 4.7 (resp. Listing 4.8) the *node-id* `C` (resp. `A`) plays the role of *subject*, `hasMRR` plays the role of *predicate* and `corollary` (resp. `lemma`) plays the role of *object*. So it can be presented in the format of a triple (*subject*, *predicate*, *object*) in exactly the same way as it is shown in the left column of Table 4.1.

Finally, for the relations annotation we use rule number 5 from Figure 4.9. First we specify a new unique (within the document) *relation-id*, then we provide a source, i.e., `src` and an anchor `anc` and finally the `relation-type`. This annotation could be presented in the same manner as it is shown in the right column of Table 4.1. For instance, the relation between two DRa nodes: `A, C`, as shown on Figure 4.8 in the form of a visible arrow, could be annotated within the *plain syntax* as follows:

```
[did@rel1 src@C type@uses anc@A];
```

Listing 4.9: The DRa relationship statement using *plain syntax*.

It is important to note that the box annotation rule (Section 4.4.1.1) is used for wrapping the chunk of text, in particular the CGa steps, if the document is already annotated with the CGa aspect. This means that the node can be only place on top of the CGa annotated step. Moreover, it is later on transformed to the XML format where the DRa node is a *parent* node for the CGa node.

In contrast, the description of the DRa node box as well as the annotation of the relationships could be located at any place in the document presented in the *plain syntax*. The choice of location of the DRa node description as well as the relation annotation is left to the author. They could be stated in the preamble of the document as well as could be stored in another document and imported to the one that they relate to.
The usage of the RDF techniques allows us to relate the attribute of the DRa node (i.e., structural or mathematical rhetorical role) to the actual node by the keyword: `about`, as presented in the examples above.

4.4.2 Concrete syntax

Documents computerised with the MathLang framework follow the XML Recommendation as a *concrete syntax*. Hence, the DRa annotation also uses the XML

format as the *concrete syntax*. The user could annotate the document using the XML schema that the MathLang framework provides. However, it is not intended to do so for various reasons, mainly because it is tedious and time consuming. Moreover, the XML representation was formulated for machines not for humans. The author can annotate the document using the *plain syntax*, which again is not intended to do so. The MathLang framework automatically transforms the computerised document from the MathLang *plain syntax* into the XML format – *concrete syntax*.

4.4.2.1 Boxes annotation

The MathLang automatic transformation from the *plain* to the *concrete syntax* is performed on every line of the *plain syntax* file. Hence it is also performed on any DRa annotation made in the document.

For example the DRa node from Listing 4.6 would be automatically transformed into the XML structure presented in Listing 4.10.

```
<dra:node xml:id="51_C">
  <cga:step xml:id="50">
  ...
  </cga:step>
</dra:node>
```

Listing 4.10: The *concrete syntax* presentation of the version of the DRa node annotation in *plain syntax* taken from Listing 4.5

In the above Listing, the XML nodes starting with the `dra:...` string belong to the DRa aspect namespace [11], whereas the XML nodes `cga:...` belong to the CGa aspect namespace. In this thesis we mainly concentrate on the transformation of the DRa aspect annotated on the *plain syntax*. Therefore, for brevity, we replace the transformation of the CGa annotation with string: '...'. More information about the transformation of the CGa aspect into XML format could be found in M. Maarek's PhD thesis [Maa07].

Each DRa node provides a unique name, which is annotated as the value of the attribute `did` using the *plain syntax*, as presented on Listing 4.6. This DRa `node-id` is transformed to a unique name id in the XML format. The `did` is converted and stored as the well-known attribute `xml:id`. The value of the `did` is transformed to the value of the `xml:id` attribute. This attribute value has to be

[11] W3C Recommendation 16 August 2006, `http://www.w3.org/TR/REC-xml-names/`

unique within the whole document which is restricted by the W3C Recommendation [12]. During the standard XML document validation with existing XML tools, we can capture information if the DRa nodes have been uniquely named within the whole document. We can also capture this on the DRa validation processing.

4.4.2.2 Boxes description and Relationships annotation

Similarly to the above transformation of the DRa nodes from the *plain syntax* to the *concrete syntax*, the MathLang framework transforms the boxes description and relationships.

For example, the role assignment from the Listing 4.8 to the DRa node is transformed and presented in the XML format as follows:

```
<dra:description about="C"
            hasMathematicalRhetoricalRole="corollary" />
```

Listing 4.11: The *concrete syntax* presentation of the version of the *rhetorical roles* assignment to the DRa node in *plain syntax* taken from Listing 4.7

This DRa node attribute assignment, i.e., mathematical or structural rhetorical role attribute, is placed in the XML file at the same location as it was added in the *plain syntax*. Although this can be changed by relocating the description of the node to another location in the *plain syntax* file, e.g., at the preamble of the document.

Similarly, the relationships can be transformed into the XML format as follows:

```
<dra:relation xml:id="rel" src="C" type="uses" anc="A"/>
```

Listing 4.12: The DRa relation taken from Listing 4.9 and transformed into the XML format

4.5 Dependency Graph

Using the DRa annotation system we can capture the role that a passage of text plays in a document. More importantly, we also capture the relationship that the role of an annotated chunk of text imposes on the rest of the document or other chunks of text. This leads to an automatic generation of a visible dependency graph for the text (see Figure 4.6), where relations between parts of text are represented

[12] W3C Recommendation 5 September 2005, `http://www.w3.org/TR/xml-id/`

by visible arrows and graph nodes have specified (but not visible) mathematical rhetorical or structural roles [KMRW07a].

From the annotated narrative aspect of a document we receive a *dependency graph* (DG) between the chunks of text in the document (e.g., see the left hand side of Figure 4.11). 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 in Section 5.3 that these dependencies give the ability to structure the skeleton of a document in the formal language Mizar (see

4.5.1 The Definition of a DRa dependency graph

As mentioned above, the MathLang user annotates the document himself. When annotating the DRa he has to annotate explicitly those passages of text that he found useful and that play some rhetorical mathematical or structural role within the document. At the same time he annotates dependencies between a number of boxes and passages of text. This, as described above, provides a *dependency graph*. In this section we describe the *dependency graph* in a mathematical way.

$G = (V, A, E)$ where $A \subseteq V \times (MR \cup SR)$, $E \subseteq V \times L_d \times V$
$V = \{n \mid n = nodeId\}$ – set of vertices
$A = \{a \mid a = (n, r) \ \wedge \ r \in MR \cup SR \ \wedge \ MR \cap SR = \emptyset\}$ – set of vertices attributes
$E = \{e \mid e = (n_{src}, \alpha, n_{anch}) \ \wedge \ n_{src}, n_{anch} \in V \ \wedge \ \alpha \in L_d\}$ – set of edges

where
$L_d = \{\mathsf{relatesTo}, \mathsf{justifies}, \mathsf{subpartOf}, \mathsf{uses}, \mathsf{inconsistentWith}, \mathsf{exemplifies}\}$ – the set of allowed labels in a dependency graph
MR – the set of MathematicalRhetoricalRoles, cf. Table 4.5
SR – the set of StructuralRhetoricalRoles, cf. Table 4.5
nodeId – a unique name/identifier given by the user while wrapping the text with boxes

Figure 4.10: The formal definition of the *dependency graph*.

The DRa *dependency graph* is a directed graph with labelled edges. We represent it as a triple $DG = (V, A, E)$ of sets, such that $A \subseteq V \times (MR \cup SR)$ and set $E \subseteq V \times L_d \times V$. The elements of set V are the *vertices* (or *nodes*, or *points*) of the graph DG, the elements of A are *attributes* of nodes and the elements of E are labelled *edges* of the graph.

Nodes of the graph are expressed in terms of names given to the wrapping boxes

around chunks of text. For example, a box annotated with letter a (or b, etc.) on the right hand side of Figure 4.1. Within the definition of the DG the node is annotated as $n \in V$.

Attributes of *nodes* are expressed in terms of *division elements* and *mathematical units*. The elements of set A are pairs $a = (n, r)$ where $r \in SR \cup MR$ is either a structural (i.e., SR) or a mathematical (i.e., MR) rhetorical role played by the wrapped chunk of text (annotated as *node* n) in the document. It is also important to note that structural and mathematical rhetorical roles are two disjoint sets $SR \cap MR = \emptyset$, where each of which is a collection of instances of the class StructuralRhetoricalRole and MathematicalRhetoricalRole respectively. The *attribute* is assigned to a *node* by using relations hasStructuralRhetoricalRole or hasMathematicalRhetoricalRole in the RDF triples. For example, we can annotate that the unique node a, on the right hand side of Figure 4.1, plays the mathematical role 'definition'. We annotate this fact using RDF triple as follows: $(a, \mathsf{hasMathematicalRhetoricalRole}, \mathsf{definition})$.

Edges of the DG graph represent relations between annotated chunks of text within the document. Moreover, each *edge* is an ordered triple, where the order of nodes in the *edge* provides an order of the relations within the document. Describing it formally, the *edge* $e \in E \subseteq V \times L_d \times V$ is a triple $e = (n_{src}, \alpha, n_{anch})$ where n_{src} is a source node of the *edge* and n_{anch} is a target/anchor node of the same *edge*. The *edges* of the DG graph are labelled α with one of the predefined DRa relations presented in Section 4.2.2.2 (or see the table in Figure 4.5), i.e. $\alpha \in L_d = \{\mathsf{relatesTo}, \mathsf{justifies}, \mathsf{subpartOf}, \mathsf{uses}, \mathsf{inconsistentWith}, \mathsf{exemplifies}\}$. The label α is annotated as a middle component of the triple and corresponds to the middle component of the RDF triple.

The above described definition of the graph is presented in Figure 4.10.

4.5.2 The automatically extracted dependency graph of a document

The MathLang DRa aspect is annotated by the user himself. The accuracy of this annotation is left to the user's understanding of the text. However, we believe that at the DRa level of the computerisation of the mathematical documents we can capture mistakes in the annotation of parts of the text.

As mentioned in Section 4.3 describing the DRa annotation process, the author performs the annotation with as little effort as possible. He simply has to wrap chunks of text with boxes, to uniquely name those boxes, to specify rhetorical roles

(either structural or mathematical) and finally to provide a number of relations between such annotated boxes. This extends the former MathLang computerised version of the document with vital information of dependencies of passages of text within the document.

At this level we can extract the annotated rhetorical structure of the document and present it in various formats. The extraction algorithm, see Section 6.3, provides one possible view of the document, mainly the *dependency graph* (DG).

In Figure 4.11 the DG for the proof of the Pythagoras' theorem which was presented and annotated in Section 4.3.

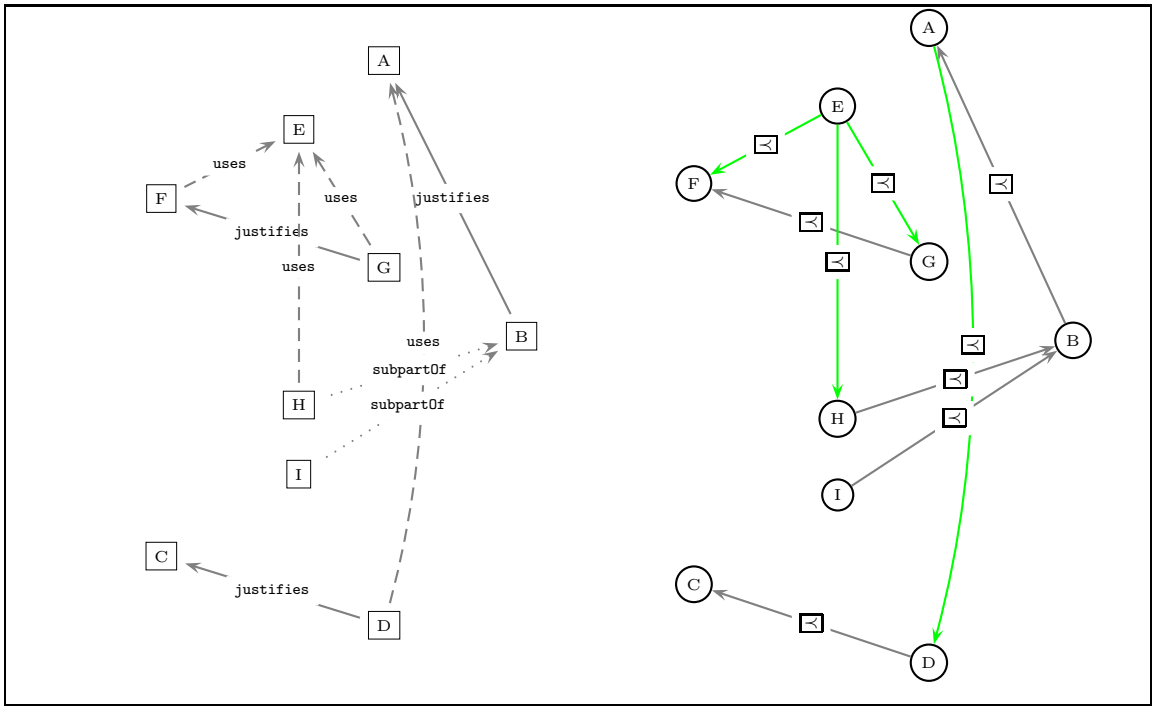


Figure 4.11: The DG and GoLP of the proof of the example of Pythagoras' theorem by H. Barendregt.
On the left hand side we have the automatically generated presentation of the *dependency graph* constructed from the input of the mathematician in Section 4.3.1 for our example of Figure 4.8. The right hand side of the figure presents automatically generated *GoLP* from the *dependency graph*.
To easily distinguish graphs, the DG from the GoLP we use different shape for nodes, mainly *square* node and *circle* node respectively.

A document's dependency graph is a directed graph with labelled edges and attributes assigned to the vertices (see Figure 4.10). The vertices (resp. attributes resp. edges) of such graph are the names of boxes (resp. mathematical or structural rhetorical roles resp. relations) specified by the user during the first (resp. second resp. third) step of the annotation of the document described in Section 4.3.1.

The left hand side of Figure 4.11 presents the dependency graph of the example of Pythagoras' theorem by H. Barendregt (see Figure 4.6). This graph consists

of (1) relations between parts of the text which are represented by visible arrows, and (2) graph nodes which have specified (but not visible) mathematical or/and structural rhetorical roles. Dependencies between the annotated chunks of text play an important role in mathematical knowledge representation. Thanks to those dependencies, the reader finds his own way while reading the text without the need to understand all its subtleties. Moreover, we will show in the next sections that these dependencies allow one to present other views on a document, and to structure the skeleton of a document in the formal language Mizar. Dependencies graphs (and their views as in Figure 4.11) are extracted automatically from the mathematicians' input in Section 4.3.1.

4.6 Graph of Logical Precedences

The above Section 4.5 presented the *dependency graph* (DG) that the mathematician has annotated during the computerisation of the DRa aspect of the document. In this section we provide different representation of the DG graph. We transform the DG into the so called *graph of logical precedences* (GoLP). We also define the *logical precedence* of mathematical relations in this Section. Moreover, we define a function that when applied to the *dependency graph* (DG) returns the *graph of logical precedences* (GoLP). We also express the algorithm for extracting the GoLP from the DG graph. All these definitions and presentations are based on the example of the Pythagoras' theorem (see Figure 4.6).

4.6.1 Logical precedences of mathematical relations

The annotation identifies and makes explicit different parts of the text, stores either the mathematical or structural or both roles of each chunk of text, and annotates the relations between recognised chunks of text (see Section 4.3.1). The use of the DRa system allows us to express relations explicitly in the computerised version of the original document. This explicit representation of relations allows to build a graph of logical precedences between different chunks of text.

4.6.1.1 What is logical precedence?

The ***logical precedence*** between two chunks of text indicates the relative positions of the chunks in a sequence of reasoning steps. These (and other) steps, contribute to the analysis of the logical correctness of the original text. *Logical*

precedence is independent of the sequential textual appearance of the chunks of text in a document. For instance, in Figure 4.8, the "Proof" (node `D`) is stated after "Corollary" (node `C`). Moreover, both nodes relate to each other, mainly node `D` is said to *justify* node `C`. However, the *logical precedence* between D and C is the opposite (see the direction of the arrow established between both nodes in the right hand side of Figure 4.11). In such case, we say that D logically precedes C.

Another explanation of the logical precedence is described in the following paragraph. In the "world" of mathematics if we want to say that a statement s is valid or states the truth, we need to give evidence or prove it beforehand. This means that the reasoning p of the statement s has to be performed and checked before s is stated. This checking of the proof object beforehand stating the statement is expressed as the logical precedence, which implies that s succeeds p.

Similarly, if we want to say that some statement a uses statement b, it could be expressed in terms of *logical precedence* as a succeeds b.

We introduce two different kinds of *logical precedences*:

- ***Strong Logical Precedence*** $\prec$: If node A has been proved by the block of reasoning steps introduced in node B, we say that A logically succeeds B and write $B \prec A$.
  Similarly if a node A uses a declared/defined symbol or a statement introduced by a node B, we say that A *follows* B and write $B \prec A$.
  Equally, if node $A1$ is one of the cases of the proof of the statement introduced in node B, we say that all the cases have to be proved before the statement in node B is said to be true or justified. In particular a case stated in node A has to be proved before the whole proof is stated. So we say that B succeeds A and write $B \prec A$.

- ***Common Logical Precedence*** $\backsimeq$: If there is a connection between node A and node B, for example both nodes can use at least one common symbol or a statement, then we write $A \backsimeq B$. This *common logical precedence* does not impose the direction of the precedence, this means that from the $A \backsimeq B$ one can get the precedence $A \prec B$ or the precedence $B \prec A$. The *common logical precedence* is similar to the equality sign "=", however statements in nodes A and B do not impose equality on any level of information.

In Section 4.2.2 we defined the DRa ontology which allows to specify the relations between recognised StructuredUnits in a document. Each such DRa relation

$G' = (V', E')$ where $E' \subseteq V' \times L_p \times V'$
$V' = \{n' \mid n' = nodeId\}$ – set of vertices
$E' = \{e' \mid e' = (n'_{src}, \alpha', n'_{anch}) \ \wedge \ n'_{src}, n'_{anch} \in V' \ \wedge \ \alpha' \in L_p\}$ – set of edges
where $L_p = \{\preceq, \prec\}$ – the set of *logical precedences* in GoLP

Figure 4.12: The formal presentation of a *graph of logical precedences* (GoLP).

between two nodes introduces logical precedence. Table 4.2 shows relations and their logical precedence in reasoning.

Relation	**Meaning**	**Precedence**
$(A, \mathsf{justifies}, B)$	A is the proof object of B	$A \prec B$
$(A, \mathsf{subpartOf}, B)$	A is a case or a part of B	$A \prec B$
(A, uses, B)	A uses a statement or a symbol of B	$B \prec A$
$(A, \mathsf{inconsistentWith}, B)$	some statement in A contradicts statement in B	$B \prec A$
$(A, \mathsf{exemplifies}, B)$	Statement in A plays a role of example for B	$B \prec A$
$(A, \mathsf{relatesTo}, B)$	There is a connection between A and B	$A \preceq B$

Table 4.2: DRa relations their meanings and *logical precedence*

4.6.2 The automatically generated Graph of Logical Precedences: GoLP

Using *logical precedences* of defined relations (see Table 4.2), one can automatically generate for a mathematical text, a *Graph of Logical Precedences* (GoLP). The right hand side of figure 4.11 shows the automatically generated GoLP for example 4.6. GoLP is a directed graph with labeled edges, achieved by the automatic transformation of the dependency graph using the transformation function *Trans* presented in Figure 4.13. In a GoLP, the direction of an edge together with the label of that edge expresses the *logical precedence* corresponding to the relation in a dependency graph from which the edge (in the GoLP) was achieved. Figure 4.12 gives the formal definition of a *Graph of Logical Precedences*.

The transformation process of the *dependency graph* DG to the *graph of logical precedences* GoLP of the annotated document is easy and simple. It consist of two actions performed on each edge of the DG graph: (1) it changes the direction of the arc of the edge, if required, (2) it changes the label for the edge of the graph. As said above, during the transformation we change the arcs of the edges drown in the DG graph to some opposite arrows connecting the same nodes as the previous relation. The arc direction transformation is done according to Table 4.4. In our example of Figure 4.11, the arrows that the direction has changed from the DG are

Graph transformation

$Trans : G_{DG} \rightarrow G'_{GoLP}$

$Trans((n, a, e)) = (n', e')$

(where $n' = Trans_{V}(n)$

and $e' = Trans_{E}(e)$)

Vertex transformation

$Trans_{V} : V_{DG} \rightarrow V'_{GoLP}$

$Trans_{V}(n) = n$

Edge transformation

$$Trans_{E} : E_{DG} \rightarrow E'_{GoLP}$$

$$Trans_{E}((n_{src}, \mathsf{relatesTo}, n_{anch})) = (n'_{src}, \backsimeq, n'_{anch})$$

$$Trans_{E}((n_{src}, \mathsf{justifies}, n_{anch})) = (n'_{src}, \prec, n'_{anch})$$

$$Trans_{E}((n_{src}, \mathsf{subpartOf}, n_{anch})) = (n'_{src}, \prec, n'_{anch})$$

$$Trans_{E}((n_{src}, \mathsf{uses}, n_{anch})) = (n'_{anch}, \prec, n'_{src})$$

$$Trans_{E}((n_{src}, \mathsf{inconsistentWith}, n_{anch})) = (n'_{anch}, \prec, n'_{src})$$

$$Trans_{E}((n_{src}, \mathsf{exemplifies}, n_{anch})) = (n'_{anch}, \prec, n'_{src})$$

(where $n'_{src} = Trans_{V}(n_{src})$ and $n'_{anch} = Trans_{V}(n_{anch})$)

Figure 4.13: The *dependency graph* transformation function.

display in 'green' color on the right hand side of Figure 4.11.

At the same time, as we change the arc direction, we assign a new label for the arc according to the same Table 4.4. The labels present the two types of logical precedences, i.e., *strong logical precedence* $\prec$ and *common logical precedence* $\backsimeq$.

Finally, let us assume that G is the *dependency graph* (DG) and G' is the *graph of logical precedences* ($GoLP$) shown in Figure 4.11 (on the left hand side and the right hand side of the Figure), respectively. Following the algorithm expressed above, the transformation function *Trans* shown in Figure 4.13 and applied to the graph G results in G'.

4.7 Automatic Analysis of the DG and the GoLP

This section provides the description of checking algorithms for the DRa annotation. The checking of the DRa is done on two levels and in two phases:

1. *The DG Level* – checking the annotation of distinct roles of recognised fragments of text and the correct usage of labels and relations.
2. *The GoLP Level* – Checking that the logical precedences in the GoLP are self-consistent, and that the GoLP is acyclic directed graph.

4.7.1 Pre-analysis of the dependency graph.

The first phase of checking catches some inconsistencies while representing the different roles of recognised chunks of text and the stated dependencies between them. For instance, if two chunks of text were annotated as "proof" resp. "axiom", and if a relation justifies is stated between them (i.e. $(proof, \mathsf{justifies}, axiom)$), the first validation stage returns an error. This error can be interpreted in two ways: first, on the relation type – which indicates that the relation might/should be different, and second on the role of each chunk of text – which roles might have been mistakenly specified.
Similarly, someone could annotate the relation "justifies" between two nodes of mathematical roles "proof" and "definition", as follows: $(proof, \mathsf{justifies}, definition)$. This annotation when checked would result in a warning either on the relation type or on the nodes mathematical roles assignment, similarly to the above description.

This validation requires additional information that is implicitly attached to each *mathematical role*. This additional information expresses if the DRa node is *provable* or *unprovable*. For instance, "lemma", "theorem", "corollary" are *provable* nodes, whereas "axiom", "definition" are *unprovable* nodes. Table 4.3 shows some *provable* and *unprovable mathematical roles*.

Provable (*by default*)	**Unprovable** (*by default*)
theorem, lemma, corollary, proposition, case,	axiom, postulate, assumption, definition, conjecture, proof, exercise, example,

Table 4.3: Some provable and unprovable entities of mathematical documents.

This checking captures other cases. Assume that one has specified simultaneously two MathematicalRhetoricalRoles for a chunk of text, for instance "axiom" and "proposition". In such a case the analysis returns a warning stating that "axiom" cannot be provable, whereas "proposition" can. Similarly, if one simultaneously states two different StructuralRhetoricalRoles for one chunk of text (e.g., "chapter" and "subsection"), the analysis will return a warning. The difference between a "chapter" and a "subsection" is that the background knowledge of a "chapter" is something like an external library for the following sections and subsections, whereas for "subsection" the context is more specific and composed of small chunks of text from the previous sections or chapters, although both "chapter" and "subsection" may use the external knowledge.

4.7.2 Checking the Consistency of Labels in a GoLP

To allow the analysis of a GoLP we have identified a number of common relational properties for *logical precedences* (see Table 4.4). These properties are used while checking the labeling consistency in a GoLP – see the following section.

Relational properties	*Common logical precedence*	*Strong logical precedences*
	$C \preceq C' \Longrightarrow C \prec C' \vee C' \prec C$	
irreflexivity	$\neg(C \preceq C)$	$\neg(C \prec C)$
symmetry	$C \preceq C' \Longrightarrow C' \preceq C$	
asymmetry		$C \prec C' \Longrightarrow \neg\ (C' \prec C)$
transitivity		$A \prec B \wedge B \prec C \Longrightarrow A \prec C$

Table 4.4: Relational properties of *logical precedences*

We build a *transitive closure* of a GoLP (using for example Roy-Warshall's algorithm [Roy59, War62]) from a dependency graph of the original document. Furthermore we check if such a built graph is the graph of a *strict partial order* (i.e., that no edge in the *transitive closure* graph has its reflexive image in the GoLP), where the *strict partial order* relation is the *strong logical precedence*.

We now give a formal definition of the *transitive closure* of a directed *Graph of Logical Precedences*. Let us take a directed *Graph of Logical Precedences* $G' = (V', E')$ where V' is a set of vertices and $E' \subseteq V' \times L_p \times V'$ is a set of directed labeled edges denoted as (v, α', w), where $v, w \in V'$ and $\alpha' \in L_p$. We denote by $\pi^{\alpha'}_{v,w} = \{v, v_0, v_1, \ldots, v_k, w\}$ a path from the initial vertex v to the terminal vertex w in G', which goes through the vertices $v_0, v_1, \ldots, v_k$, where $k \in \mathbb{N}$, and where each edge between two vertices is (v_i, α', v_{i+1}), where $i \in \{0, \ldots, k-1\}$.
A *transitive closure* of graph G' is a graph $G^+ = (V', E^+)$ such that E^+ contains an edge (v, α', w) if and only if G' contains a path $\pi^{\alpha'}_{v,w}$.

We illustrate the analysis of consistent labeling on the GoLP based on our example shown on the right hand side of Figure 4.8. Take the nodes E and F, and the edge (E, α', F), where $\alpha' \in L_p$ (see Figure 4.14). In the transitive closure of our GoLP we have two paths that form the edge (E, α', F): (1) a direct path $\pi^{d,\alpha'}_{E,F} = \{(E, \prec, F)\}$, and (2) an indirect path $\pi^{ind,\alpha'}_{E,F} = \{(E, \prec, G), (G, \prec, F)\}$. The direct path is labeled with a *strong logical precedence* symbol $\prec$, denoted as $\pi^{d,\prec}_{E,F}$. When evaluating the label of the indirect path $\pi^{ind,\alpha'}_{E,F}$, we have to take into account the relational properties of the *logical precedences* of Table 4.4. In our case, we use the transitivity of *strong logical precedence* $\prec$ between the two edges of the path $\pi^{ind,\alpha'}_{E,F}$. From this, we obtain the labelled indirect path $\pi^{ind,\prec}_{E,F}$, which has the same label as the direct path $\pi^{d,\prec}_{E,F}$. We conclude that the edge (E, α', F) in the Graph

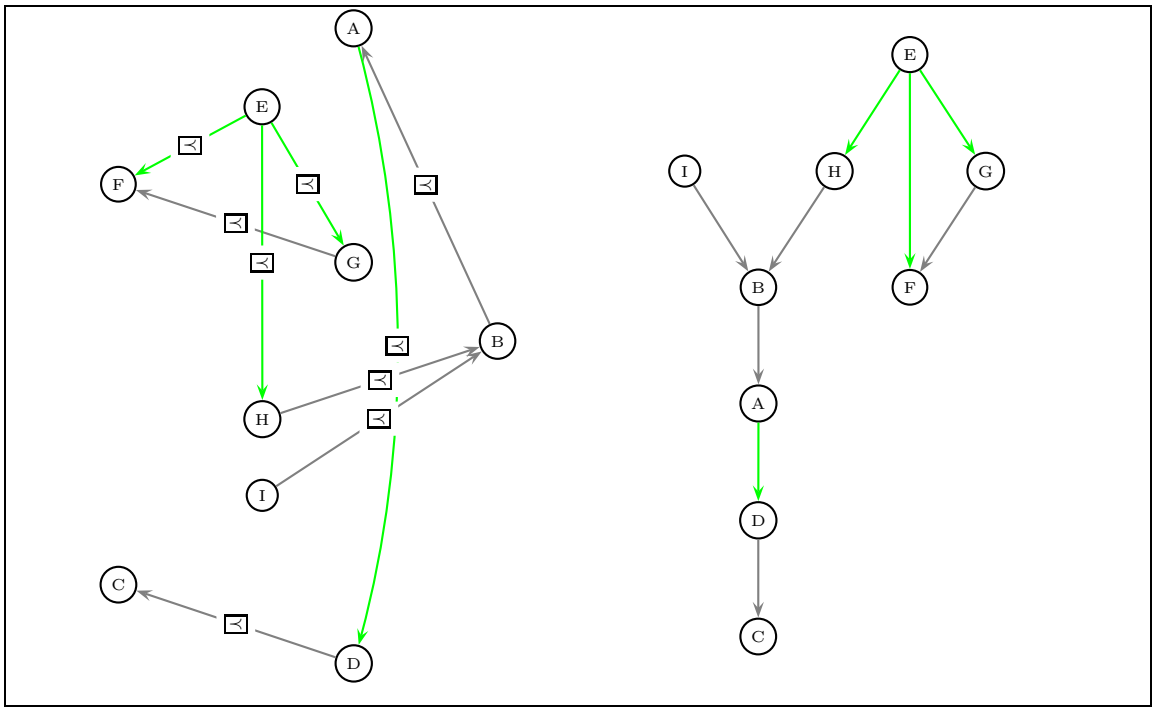


Figure 4.14: The GoLP of the proof of Figure 4.6's example represented in two different ways.

On the left hand side we have the automatically transformed GoLP from the DG of the example of Figure 4.8. The right hand side presents a different view of the same *GoLP* with the *precedences* show and preserved from the left hand side, also together with the colors of edges. Edges labels are stripped out for the viewing and clarity purposes.

of Logical Precedences (GoLP) is labeled consistently.

Labeling consistency validation is performed on each existing edge in the *transitive closure* of GoLP built from a dependency graph of the original document. Once we go through the whole checking of the graph we can say that the GoLP is valid according to the consistent labeling.

4.8 Conclusion

In this chapter we presented our main contribution of the thesis. Mainly we described another aspect of mathematical documents, which is part of the MathLang framework. The aspect is oriented toward capturing narrative structure of any mathematical document. In the first section of this chapter we provided an overview of existing tools for computerising rhetorical structure of any ordinary document. We presented and roughly described those system with number of toy examples. Furthermore, we explained why the DRa aspect was born, and what we get from it. We discussed the formal aspect of the DRa, its web-technology driven development and its ontology. We also illustrated the process that the user needs to perform to annotate his document with the DRa information on top. We explained the *plain* and *concrete* syntaxes following by a number of small examples. Furthermore, we presented the Dependency Graph and our formal definition of it. The DG is annotated by the user himself, during annotation of relations among distinguished parts of document. Similarly, we introduced *logical precedences* and the Graph of Logical Precedences of a document. The GoLP is automatically generated graph from the existing DG graph of a document. We expressed also the transformation function θ :DG$\rightarrow$GoLP. Finally in the last section we described the pre-analysis of the DG annotation performed by the user. Going further we discussed possibility of GoLP checking.

Chapter 5

Gradual Computerisation

In this chapter we present our approach to gradual computerisation of mathematical documents. Generally speaking our gradual computerisation starts from CML via MathLang and its aspects, the via Mizar Formal Proof Sketch towards full formalisation in Mizar. The same steps from the MathLang aspects could be followed to achieve formalisation in other *proof systems*. Other students of Prof. F. Kamareddine and Dr. J.B. Wells, R. Lamar and C. Zengler are doing research to extend MathLang aspects to achieve formalisation in other *proof systems* (R. Lamar towards Isabelle and C. Zengler towards Coq).

Section 5.1 presents three different paths that a mathematician/user can follow to achieve the same goal – the formalisation in the Mizar system. The first path can be achieved by direct formalisation of an original CML document into the Mizar language. The second path is done via Mizar FPS. The last path, which is our approach, the MathLang philosophy and the subject of this dissertation, is done from the CML via MathLang to Mizar FPS and finally towards full formalisation in Mizar. In the same Section 5.1 we briefly compare these three paths.

Section 5.2 presents a computerisation process that one has to do while following our proposed formalisation path. We have chosen the "Pythagoras proof of irrationality of $\sqrt{2}$" example written by G.H. Hardy and E.M. Wright [HW80]

In Section 5.3 we describe hints and rules on how the narrative features captured in the MathLang Document Rhetorical aspect are used to build the skeleton of a Mizar document. The next Section 5.4 presents transformation hints from the MathLang CGa/TSa annotation into the Mizar formula level. We describe in details which mathematical identifiers captured by CGa correspond to their formal counterparts in Mizar. We also describe transformation hints for building steps from MathLang CGa/TSa annotations to the Mizar formula level. Finally we give

a short comparison between all three formalisation paths that are described in Section 5.1.

5.1 Formalisation Paths

There exists a number of ways in which a user can formalise a mathematical document using the Mizar language and the Mizar system. The possible three approaches are sketched in Figure 5.1.

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 5.1 where all the levels at step ⓐ are done by the mathematician and where the sub-path ⓓ-ⓔ is done by either the mathematician having some previous formalisation experience in Mizar or by the Mizar expert.

Our proposed approach has a number of advantages:

- It gives a better view at the process of computerisation and is useful to understand all the steps that a user has to perform to fully formalise a mathematical document.
- 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.
- As expressed in the description of Figure 5.1, different formalisation paths involve different levels of the expertise required by the user.

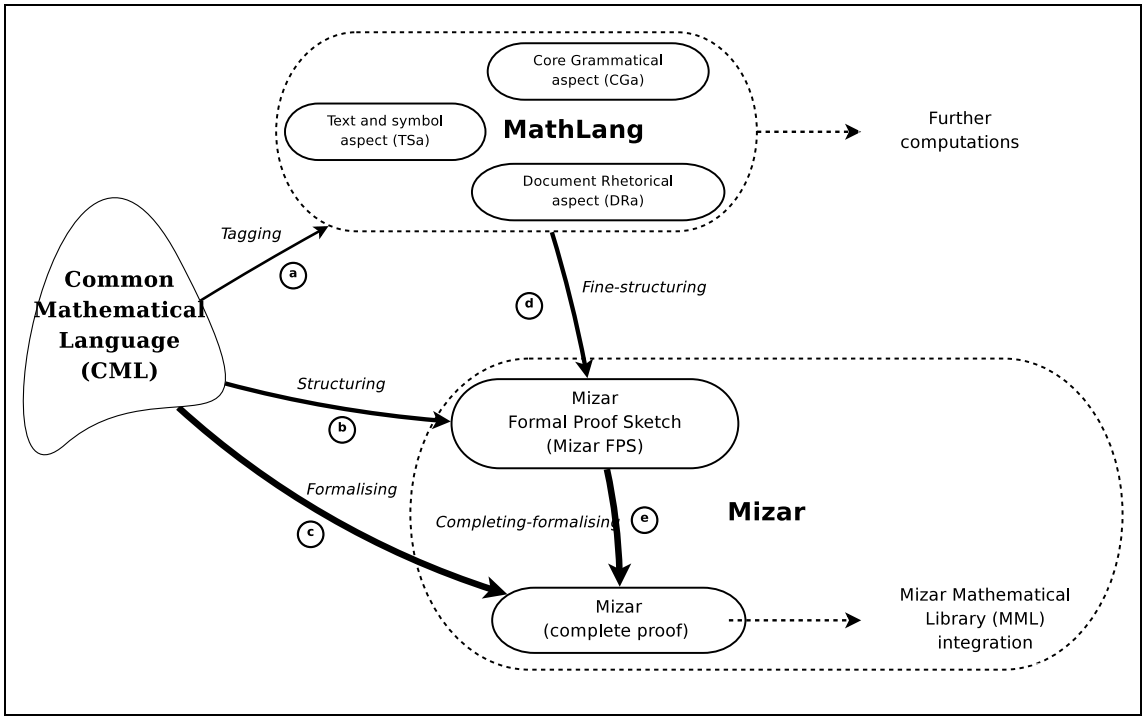


Figure 5.1: The computerisation/formalisation paths from CML to Mizar.

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.

5.1.1 The direct path from CML to Mizar – (© of Figure 5.1).

The direct formalisation path ©, as shown in Figure 5.1, starts from an original CML document and goes a long way to Mizar. The approach to do formalisation in Mizar from the CML has been described in "nine easy steps" by F. Wiedijk [Wieb]. Although, writing a Mizar article is rather straight-forward (as written by F. Wiedijk in [Wieb]), the mathematician/user who wants to do formalisation of the CML text, needs to be a specialist in the Mizar system, especially in MML and its search engines: MML Query or the `grep` tool. The reason for this is that the most difficult part of writing Mizar articles is finding what is needed to do the formalisation inside the MML. Basically, the Mizar user needs to interpret the original CML text, to find the meaning of each part of the document, and to present it in a formal way. This task however, requires a knowledge of Mizar, its language structure, constructors, patterns etc.
Furthermore, the user's expertise needs to encompass both mathematics and computer science. In fact, even if a Mizar *Article* resembles a CML text, the Mizar language is much more closer to declarative programming languages (e.g. Pascal).

Moreover, CML texts can be ambiguous, and the user needs to find and clarify those ambiguities when formalising the text. The improvements to the CML text in terms of solving its ambiguities will make the formalisation easier from the very beginning. Therefore it is highly important to understand completely the original CML document.

The user has a choice how he wants to do the formalisation. One of the possible ways is to first present parts of the text in the Mizar language and then carry on the formalisation of those parts to meet fully correct and formalise part of the original document. Another way is to start from a single statement and look in the MML for knowledge that allows the presentation of this statement in the Mizar language and then rewrite this statement in the Mizar language. If that statement is accompanied with the proof, maybe the user could fully formalize the proof and move forward to translating the rest of the document into Mizar.

Although the Mizar user has a choice, as expressed above, 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.

5.1.2 The path from CML to Mizar FPS to Mizar – (ⓑ-ⓔ of Figure 5.1).

The second possible way to do formalisation in Mizar is presented as path ⓑ-ⓔ of Figure 5.1. To perform such formalisation the user or mathematician still needs to be a Mizar specialist. This path also requires similar amount of knowledge as the user who follows the direct formalisation path ⓒ to Mizar.

The difference 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, theorem or proof. Although such a choice is possible, the Mizar FPS is not meant to do that.

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 errors `*4` or `*1` by the Mizar system). Such process is iterative and requires to do some changes of already edited/translated document. The user starts to fill up the holes in the Mizar FPS and in majority of cases it happens that the user has to change the first instance of the Mizar FPS representation to be able to eventually finish the full formalisation.

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 in a direct formalisation path from CML to Mizar (path ⓒ of Figure 5.1).

There are number of advantages to following such a formalisation path. The user has a choice of finishing the full formalisation for some parts of the document and leaving other parts as partially formalised, for instance those that the user consider as trivial or others that the user has difficulties formalising. Someone could argue that this could be done as well when following the direct path to Mizar. However, when doing formalisation throughout the Mizar FPS we can be more assured that the actual formalisation more accurately resembles the original CML document then to the direct formalisation from CML to Mizar. The reasons for this are the Mizar FPS "idea", "design", "definition" and nature, see 3.8. This close resemblance to the original CML document is a very important issue when doing

formalisation and can be called the "reliability criteria"[1]. If the formalised version of a CML document follow this "reliability criteria" the reader of the formalised document can easily find a way through the formalisation, and more importantly can easily compare the formalisation to the original document.

Although the goal, when following this "two steps" formalisation path ⓑ-ⓔ, is to achieve full formalisation, the user who is formalising the document, has a choice to leave some parts partially or not formalised at all. The further formalisation after the Mizar FPS presentation of the original document could be carried out by a more experienced Mizar specialist. This possibility and the "reliability criteria" allow an inexperienced Mizar user to still be able to do some formalisation of the original document.

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 user and a bit easier and more importantly the formalisation follows the "reliability criteria".

5.1.3 The path from CML to MathLang to Mizar FPS to Mizar – (ⓐ-ⓓ-ⓔ of Figure 5.1).

The last path ⓐ-ⓓ-ⓔ presented in our Figure 5.1, is done starting with CML text to MathLang, then to Mizar FPS and finally to full formalisation in Mizar. The first glance at that path gives a rough concept/objective of the MathLang project. Generally speaking, we believe that the formalisation process could be divided into a number of small steps, where each step could be done independently from another, but more importantly, each of those steps will capture a different aspect of the original document, and at the same time it will refine the formalisation which eventually will lead to a full formalisation.

Let us explain the proposed formalisation path. The first part of the path (ⓐ) is done by a mathematician, who does not require a lot of MathLang and computer/programming knowledge when annotating the text with CGa grammatical categories or when assigning the relationships between different parts of the text and the mathematical or structural rhetorical roles that different mathematical entities play. The mathematician simply reveals his understanding of the text. This simple annotation process gives some advantages:

- It makes explicit all the identifiers used in the text together with a number of arguments and their weak input and output types.

[1] "It is of uttermost that the formal version of a piece of mathematics covers exactly the intended contents of the original CML-version" by R. Nederpelt in [Ned02]

- It resolves some or even most of ambiguities of the text.
- It provides a fine structure of the document which can be later on 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 could be finished, although it could be carried out towards further steps in the formalisation.

Such a transformed document using MathLang aspects could be translated to Mizar FPS. The experienced Mizar user or specialist takes the tagged document within MathLang and transforms it into Mizar FPS (part ⓓ of the path). Here, the user needs to have previous Mizar experience and to have the same Mizar knowledge as in the direct path and the one via Mizar FPS.

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.

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.

5.2 An example of Computerisation Path Following MathLang Approach

In this section we describe a full formalisation process following the MathLang formalisation path. We based this process on a simple example of the "Pythagoras Theorem" by G.H. Hardy and E.M. Wright. We recall the original version (as already presented in Section 2.1) and show it on Figure 5.2.

Theorem 43 (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 $(a, b) = 1$. □

Figure 5.2: Pythagoras' proof of the irrationality of $\sqrt{2}$ by G.H. Hardy and E.M. Wright [HW80] as presented by F. Wiedijk in [Wie06]

5.2.1 Annotation of CML with MathLang CGa and TSa

The actual path starts from the original CML document, presented in Figure 5.2. Because the document that we have chosen to present the formalisation path was already written and published, the user actually wants to "translate/transform" that document into the full formalisation. However, we aim that our proposed path is suitable for both: the transformation of existing mathematics into formal "proof systems" formalisation, as well as the authoring of a new mathematical documents.

As a first step the user has to take the original document and rewrite it in TEXMACS (the scientific editor) or if the user has already written the document in TEX he can import it from the TEX sources into TEXMACS. Figure 5.3[2] presents the original document from Figure 5.2 in TEXMACS.

[2]Please note that the equation "$a^2 = 2b^2$" has been in-lined compared to the original version shown in Figure 5.2. This is due the fact of boxing model used by TEXMACS and MathLang annotation which does not allow to freely annotate equations with MathLang boxes. This bug has been reported to the TEXMACS developers.

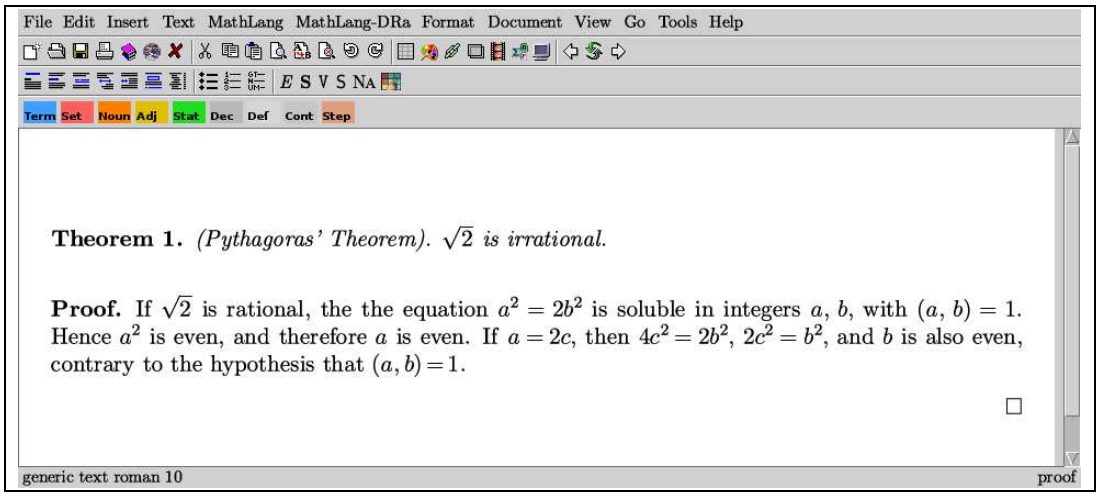


Figure 5.3: The TEXMACS presentation of Figure 5.2

The TEXMACS tool has been chosen as a MathLang preferred editor for a number of reasons. First of all TEXMACS provides lots of capabilities and features. Especially it supports a "modern approach" ("What You See Is What You Get") to writing documents in th form of a presentation layer instead of a "programming language" style.

Moreover, TEXMACS provides features like: import and export from and to a TEX/LATEX file. More importantly it has a built in easy plugin engine and supports "proof systems" including Computer Algebra SystemCAS and proof assistant via such plugin engine. We created the MathLang plugin for TEXMACS to provide mathematicians with a user-friendly editor for MathLang. The plugin also gives a complete freedom in annotating mathematical documents with the MathLang *aspects* (CGa, TSa and DRa).

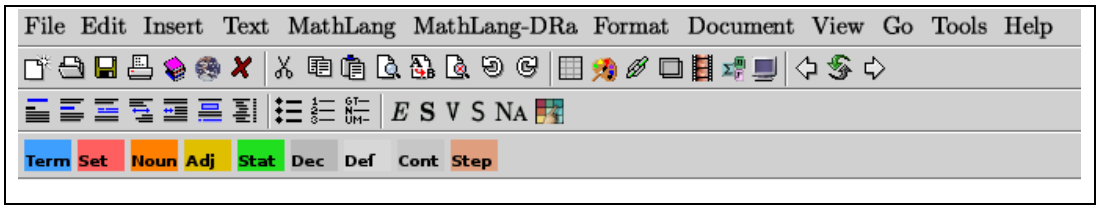


Figure 5.4: The MathLang plugin bar in TEXMACS.

When the document is already typed in TEXMACS the user can perform the annotation of the MathLang grammatical roles using the MathLang plugin. The plugin supplies several ways for annotating TEXMACS document with CGa and TSa elements. One can access CGa and TSa annotation functions via a "MathLang" menu (see Figure 5.5), via an icon bar (Figure 5.4) and via keyboard shortcuts.

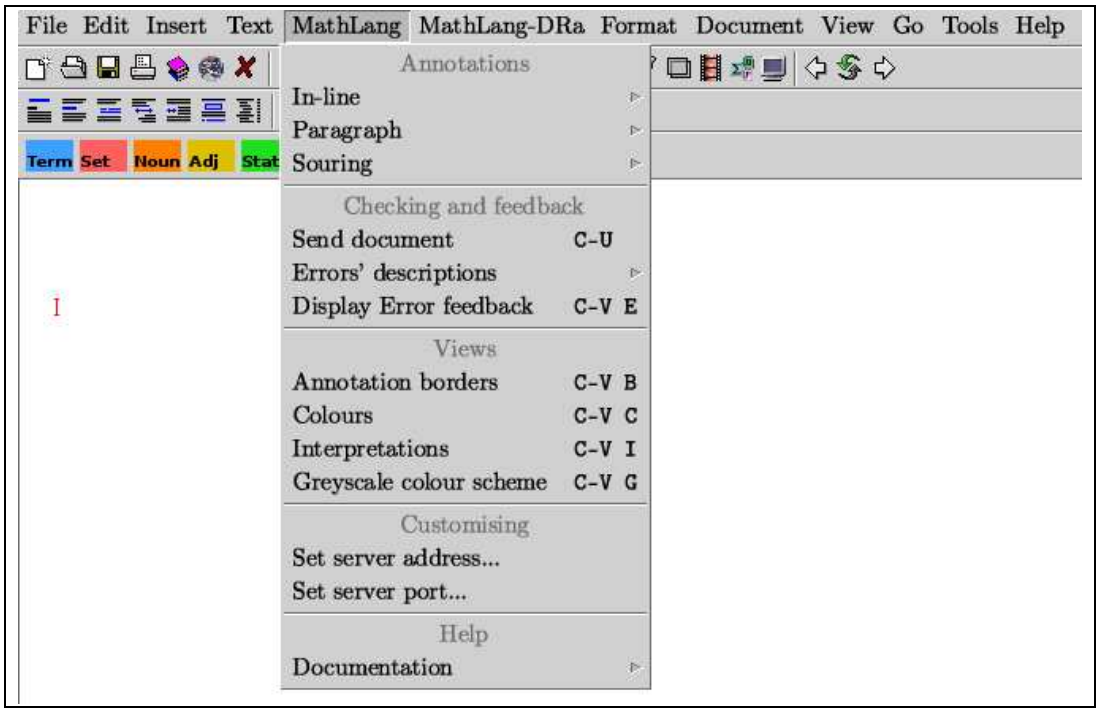


Figure 5.5: The MathLang plugin menu in TEXMACS.

The MathLang plugin for TEXMACS has two menus for MathLang annotations. The first one, called MathLang is mainly oriented for CGa and TSa annotations as well as for checking and propagating different views of the MathLang annotated document. This menu has been described in more details in M. Maarek thesis [Maa07]. Whereas the second menu, called MathLang-DRa, is oriented, as the name suggests, to DRa annotation and consists of different sections which is described in Chapter 6.

The first thing that the user has to do is to perform the annotation of the CGa and TSa on top of the already computerised document. To illustrate the annotation process with MathLang's aspects using the TEXMACS and MathLang plugin, we use the main theorem statement of our example (Figure 5.3.

We identify at this statement the grammatical role of each element of the text:

- term for "2" and "$\sqrt{2}$"
- statement for "irrational"
- definition for entire statement.

The user indicates to each element its grammatical category (CGa role) by wrapping it with coloured boxes following our colour coding system of Figure 2.6. The user has to also provide an internal CGa identifier interpretation for each annotated element.

The annotation process usually follows the set of rules:

1. wrap a fragment/element of a text with a box,
2. choose the CGa grammatical role by using either the MathLang plugin menu (see Figure 5.5), or using an icon bar (Figure 5.4) or using keyboard shortcuts,
3. provide an interpretation of the CGa annotated element, (as indicated in the left side of Figure 5.6),
4. finally, introduce the interpretation identifier in the preamble of the document together with its input and output grammatical role (for example: `sqrt(term): term` or 

, which means that the grammatical role of the argument and the output is **term**).

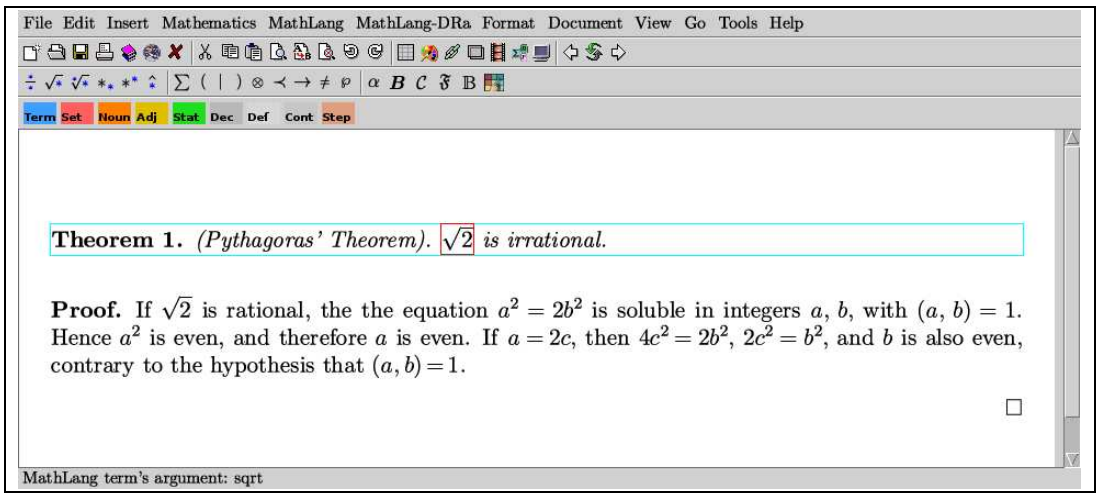


Figure 5.6: The MathLang annotation process.

By following the annotation process rules the user eventually achieves the main part of the first step (path ⓒ) of our formalisation path as shown in Figure 5.1.

As mentioned before, the MathLang CGa and TSa annotation process does not require any expert knowledge from the user. Moreover, we believe that the MathLang plugin and editor can be used even by pupils in college. The only knowledge required to successfully annotate a mathematical document with MathLang's CGa+TSa aspects is the ability to de-construct statements into elements and identify the grammatical role for each element.

Finally, the MathLang CGa+TSa version (Figure 5.7) of the original example (Figure 5.3) can be checked by the MathLang CGa checker. This has been discussed in more details in M. Maarek thesis and is not the subject of this thesis.

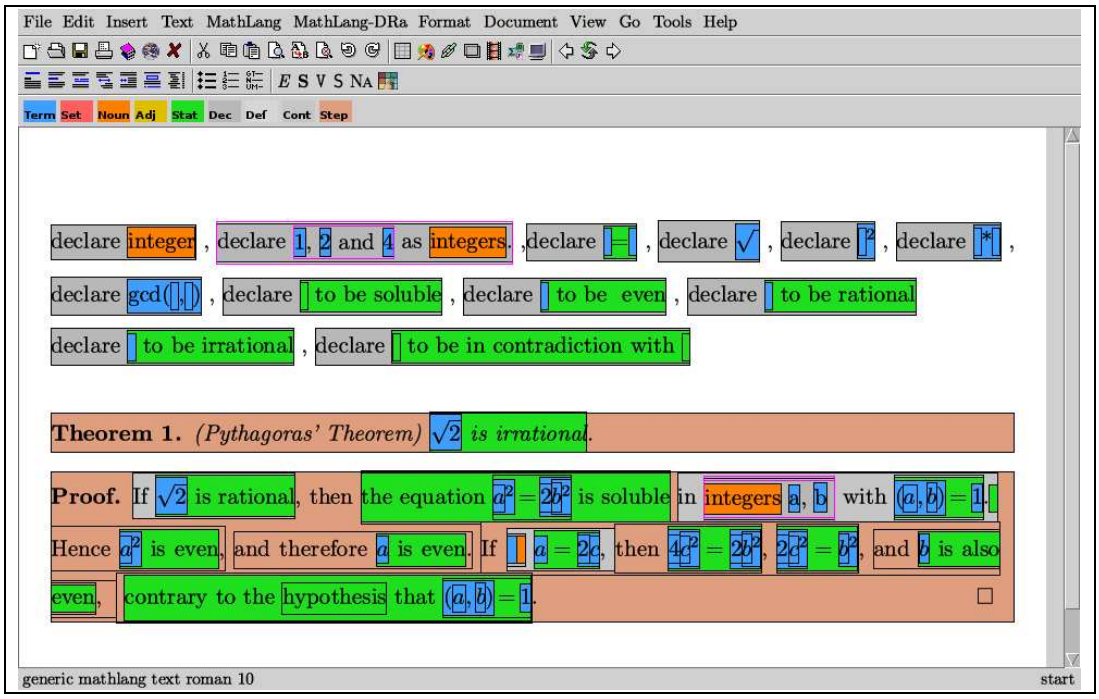


Figure 5.7: The MathLang CGa+TSa presentation of the original document from Figure 5.3.

The final presentation of the MathLang CGa+TSa computerised document consists of two parts:

1. The MathLang preamble/declarations which indicates input and output types of the interpretation of each recognised document element.

2. The main CML text computerised with grammatical roles associated to each element of the document,(see Figure 5.7).

It is important to note that the MathLang plugin has few additional features which can make the MathLang framework more attractive to potential users. The plugin can provide a number of combinations of views for the same computerised document:

1. a standard TEXMACS view, with hidden MathLang annotation (see Figure 5.3),

2. a version with colours and boxes (boxes with borders) – the default view (see Figure 5.7)

3. a version with boxes without colours (see Figure A.3 in Appendix A.2)

4. a version with the MathLang CGa internal interpretation (see Figure A.2 in Appendix A.2)

The document can be also presented in the XML view, the default internal representation of MathLang documents, and in a plain syntax (see Listing A.1 for the *plain syntax* version).

5.2.2 Refinement of the MathLang CGa + TSa computerised text with DRa

At the previous stage, the original CML document (see Figure 5.2) is computerised with the MathLang CGa and TSa. The next part of computerisation involves annotating the MathLang DRa aspect on top of the already computerised document. By annotating the DRa aspect we refine the structure of the MathLang CGa+TSa encoding.

At this stage the user identifies chunks of text that play an important role within the document. The importance of a block of text within the whole document depends on the user's interpretation and is left to his decision. In our example, the MathLang CGa+TSa version (see Figure 5.7) of the original document (see Figure 5.2), the user might identify three blocks that are worth distinguishing and wrapping with boxes: 1. theorem, 2. proof of the theorem, 3. preamble. We can annotate these blocks by using the MathLang plugin. Similarly to the previous annotation process, the user can access DRa annotation functions via the "MathLang-DRa" menu (see Figure 5.8) or via keyboard shortcuts.

The MathLang-DRa menu and TEXMACS implementation of such part of the MathLang plugin is described in Section 6.2, therefore we do not go into further details here.

Let us annotate the identified parts of the document. First we annotate theorem by selecting the whole statement including the label "Theorem 1". Then we choose from the MathLang-DRa menu the "DRa paragraph annotate..." part. This will first prompt the user to enter the "DRa structural role" in the bottom bar of the window (see Figure 5.9), then will ask for the "DRa mathematical role" and finally for the "Unique loci" which is a unique identifier in the whole document for all the DRa annotated blocks. Following such procedure we annotate the rest of the identified blocks: the proof and the preamble. The final version of blocks' annotation is shown in Figure 5.10. As we can see on the Figure, the DRa annotation view presents also the interpretation given by the user to each block. It is

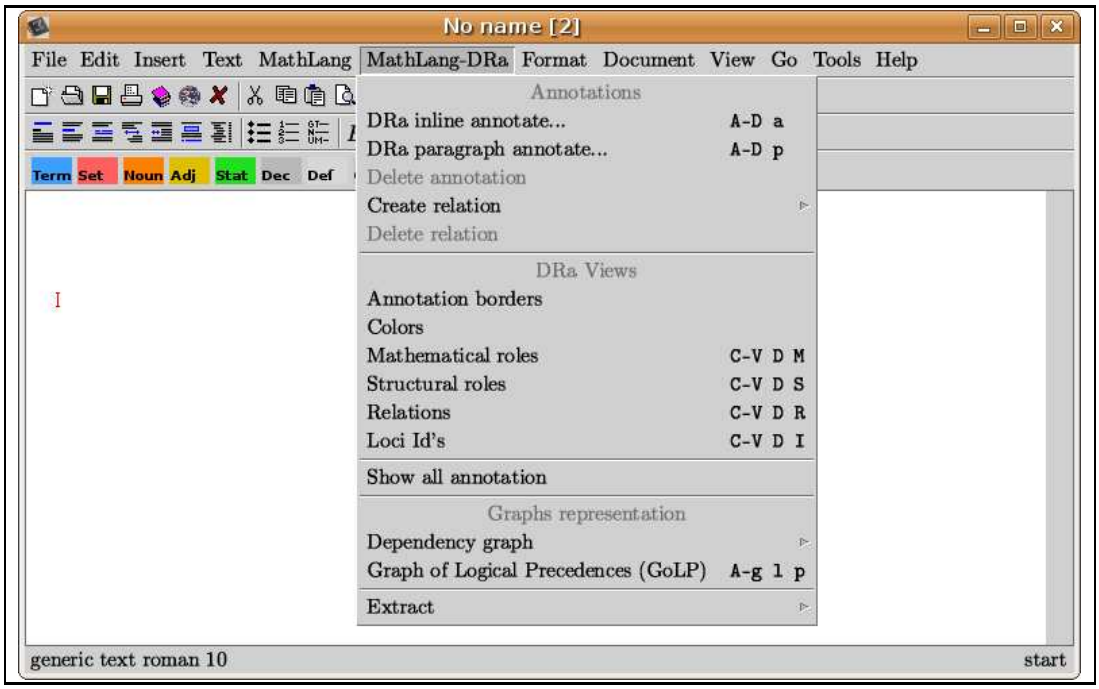


Figure 5.8: The MathLang plugin DRa menu in TEXMACS.

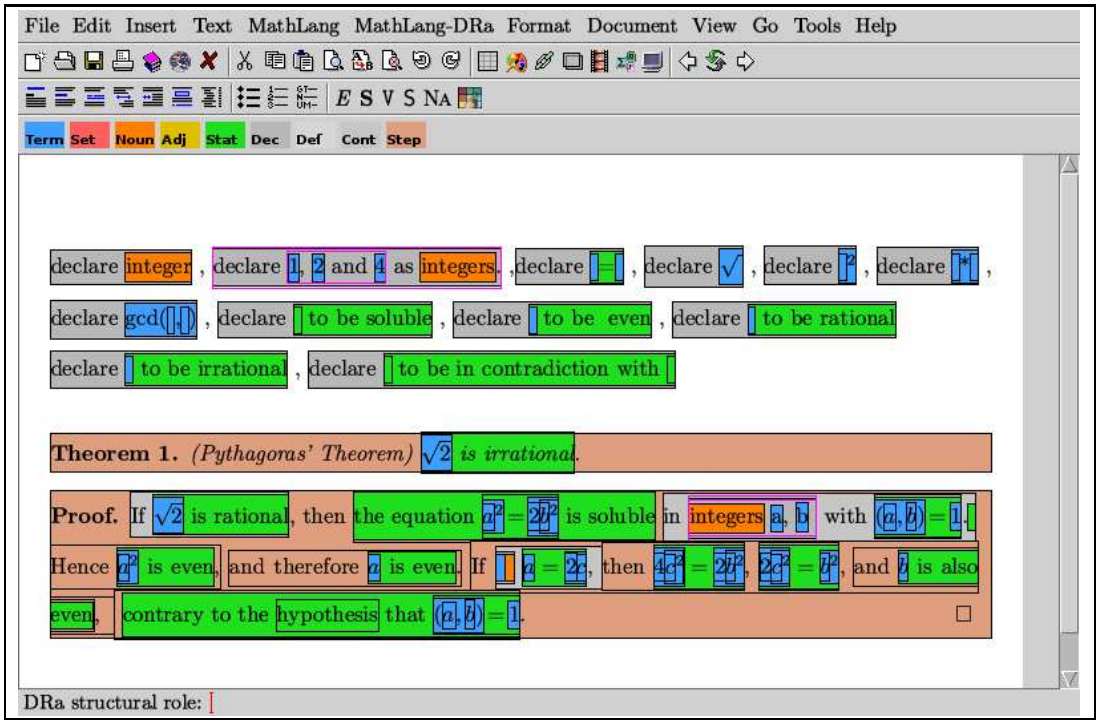


Figure 5.9: The MathLang DRa annotation of a block around the "theorem".

placed in the following order: 1. structural role, 2. mathematical role, 3. unique identifier. The interpretation could be seen as being similar to the one provided by the user for each element of the text while annotating "grammatical roles" with MathLang CGa+TSa.

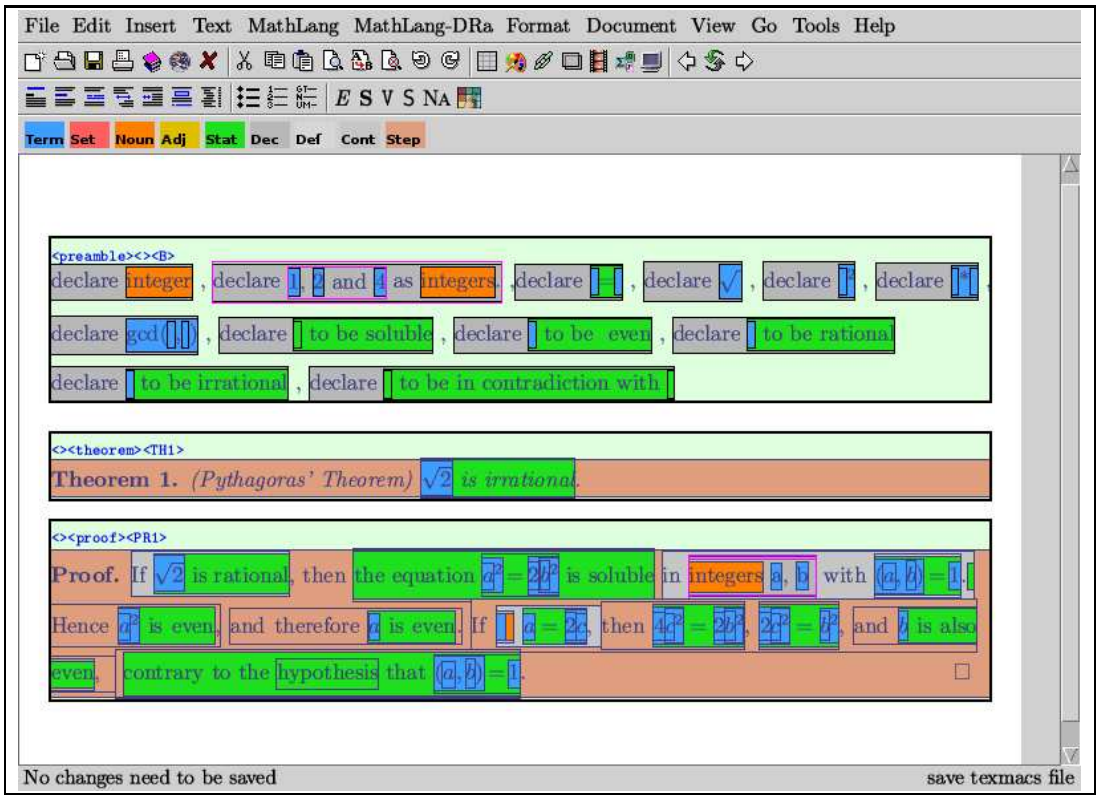


Figure 5.10: The MathLang CGa+TSa computerised document refined with DRa blocks annotation.

Such annotated document contains DRa annotations around identified important chunks of the text within the document. At this stage the user could leave the MathLang DRa annotation process. However, the purpose of the DRa is not only to identify the significant parts of the document, but also to provide and annotate relationships between these parts. The DRa aspect introduces 6 pre-defined relations (i.e., relatesTo, justifies, subpartOf, uses, inconsistentWith, exemplifies, see Section 4.2 for more explanation) that could be reused to annotate the DRa relationships in a document.

In our example we can identify at least one relation. Mainly, we can annotate that the "proof" (a box annotated with the DRa block and specified mathematical role "proof", and provided unique identifier "PR1") justifies "theorem" (a box annotated with the DRa and specified mathematical role "theorem", and provided

unique identifier "TH1"). We annotate such relation by accessing the MathLang-DRa menu and choosing the option: "Create relation → justifies...". This will prompt the user for the "source" and "target" of the relation. The "source" and "target" are specified by unique identifiers provided during block annotations of the MathLang DRa, which are in our example "PR1" for the "source" and "TH1" for the target. The relation annotation is entered in the document, wherever the mouse pointer was positioned while the relation was annotated and is not restricted to any specific location. However, it is highly recommended to place all DRa identified relations in one location in the document so they can be easily found and changed, but this is left to the user's decision. We placed the relation below the preamble block and above the main document part (see Figure 5.11).

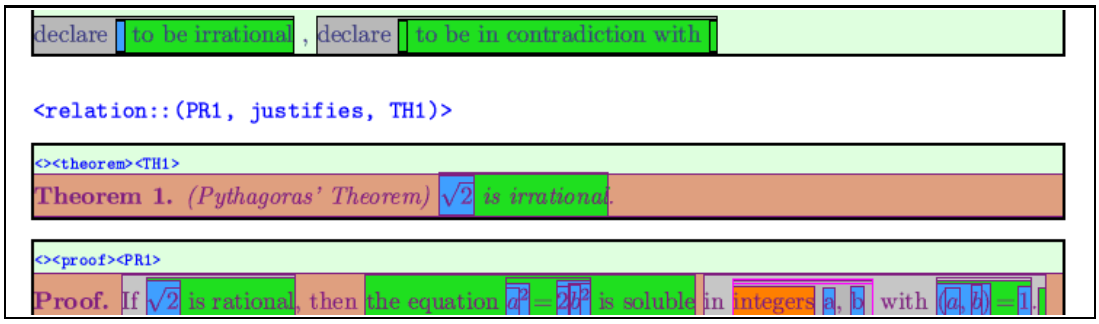


Figure 5.11: The MathLang CGa+TSa+DRa computerised document refined with DRa relations annotation.

Although, our example has only one annotated relation, the relation helps us to identify the connections and order between chunks of texts within the document. Someone could say that the annotation of the DRa is not really useful, and probably this toy example does not need the DRa. However if one wants to computerise a big CML document or part of a book like [Lan51] by E. Landau, the DRa annotation is very useful and provides a graph of relationships between chunks of text within the document.

At this stage the MathLang computerisation of the original document finishes. The user has annotated the document with all three MathLang aspects: on the first stage with the CGa+TSa aspects, and later with the DRa aspect, all happening within the first step ⓐ of our formalisation path (see Figure 5.1). The user can also change the annotation order and starts the computerisation with the DRa aspect first and later follows with the other two aspects.

5.2.3 Transformation of the MathLang document into Mizar FPS skeleton and finally to correct Mizar FPS

This step of our formalisation path takes the MathLang computerised version of an original document and transforms it into a Mizar FPS. The process here is not straightforward and requires quite a decent knowledge of the Mizar language, the Mizar system and the MML itself. More specifically, in this section we will present some pattern that a user can follow when trying to achieve a partial formalisation of a document in the Mizar language.

A user who wants to write a Mizar document or wants to transform it from the MathLang computerised version, needs to use some kind of text editor. The user is not restricted to use a specific editor tool, however we recommend to use the Emacs text editor due to the existence of a "Mizar Mode for Emacs" built by J. Urban [Urb06]. Such "Mizar Mode" provides a colour highlighting for the Mizar syntax, as well as very easy access (either via the Mizar menu or via keyboard shortcuts) to the Mizar system and tools to process and verify a Mizar Article.

As a first step in the transformation we reuse the MathLang DRa annotation of the document to build the Mizar FPS skeleton. Figure 5.10 shows the document with highlighted and annotated blocks: "preamble", "theorem" and "proof". The layout of these blocks corresponds directly to the Mizar FPS skeleton layout presented in Listing 5.1.

```
environ
 :: The Mizar environment which effectively is counterparts of
 :: MathLang preamble elements in the Mizar Language.


begin :: Proof of the Irrationality of square root of 2,
      :: original approach by Hardy and Wright.

theorem TH1:
  :: Theorem statement required.
proof
  :: Proof of the theorem statement.
  thus thesis;
end;
```

Listing 5.1: The Mizar Article skeleton of the MathLang computerised document from Figure 5.10

Once this is done, the user can run the Mizar system to verify the document. Mizar would respond with an error (as shown in Figure 5.12), which states that

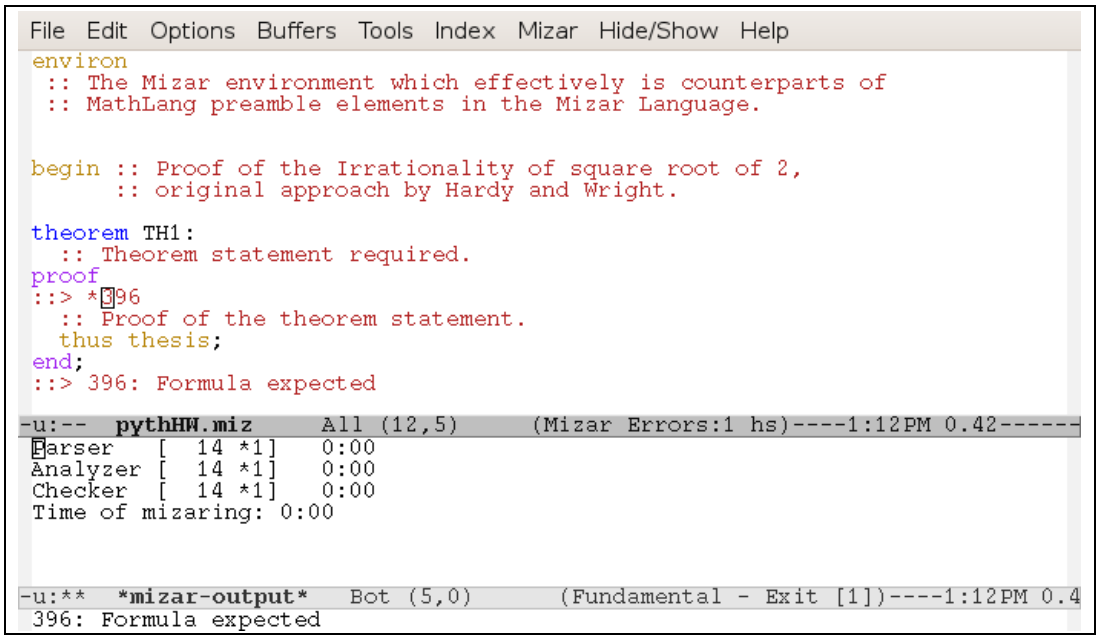


Figure 5.12: The Mizar FPS document skeleton transformed from the MathLang computerised document.

a Mizar formula is expected after the keyword *theorem* instead of the comment: "Theorem statement required". Please also note that the Mizar FPS skeleton does not contain any label after the word *proof* ("PR1" as shown in Figure 5.10) – the Mizar language does not support labels for proof blocks (except for the special Mizar proof structure – "proof by cases"). A theorem in Mizar requires a proof which must be placed straight underneath the theorem statement (see the Mizar language syntax[3]).

Now the user needs to translate the theorem statement to the Mizar language. We start the transformation of the statement by searching the MML to find the definition of the square root "$\sqrt{}$" function and symbol. In the MathLang preamble we provided the interpretation of the identifier "sqrt" as a function and identifier, which takes one argument of type **term** and returns the same type. We can assume that a similar approach and representation of the identifier is placed in MML, although such assumption is not always true. So we look up the MML vocabulary files to find the name of an article in which the "sqrt" symbol was introduced. We can do it by either using the Mizar tool in the command line (i.e., "`findvoc sqrt`") or by using the Mizar menu in Emacs (by selecting "Mizar → Voc. and Constr. Utilities → Findvoc", which effectively calls the command line tool "`findvoc`").

[3] The Mizar syntax can be found in a file `syntax.txt` in any installed version of the Mizar system.

The name of the article in which the symbol was introduced is SQUARE_1. We place this article in the "environment" *vocabularies* directive of the Mizar article. We can also assume that the "sqrt" was introduced within the same article SQUARE_1. Hence, we can find a functor definition (a function in Mizar) which introduces the "sqrt". This functor definition also provides syntax and layout of the formula, that can be used to present our theorem in the Mizar language.

As a next step, the user needs to identify a definition of the "irrationality" in the MML library. Again, we believe that the MathLang representation and interpretation of the word "irrational" can help to find a proper counterpart in the Mizar language. The word "irrational" has been introduced in the MathLang preamble as an adjective, i.e., we assigned the **adjective** grammatical role. Similarly to the previous case of the identifier "sqrt", we can use the same command to find a vocabulary article in which the adjective "irrational" has been introduced in MML. Adjectives in Mizar are represented as an *attr* and the adjective "irrational" was introduced in the article IRRAT_1. Moreover, the construction and presentation of the original CML version of the theorem looks identical to the Mizar representation, which is as follows:

```
sqrt 2 is irrational
```

However, this is not a surprise, because the Mizar language aims to mimic the Common Mathematical Language. It actually looks very similar to the MathLang CGa *plain syntax* representation (see Listing A.1 in Appendix A.2).

Now the user needs to fill in the missing parts of the environment. The first thing to do is to find out what constructors are used by the Mizar system to "understand" the meaning of the expression. We can find this information by using the Mizar command line tool *constr*. Once the command[4] is run, as a response we receive 4 articles that have to be placed in the environment directive *constructors* (NUMBERS, XXREAL_0, XREAL_0, SQUARE_1). Such filled in Mizar environment is still not sufficient enough to provide a full knowledge for the Mizar system to translate and understand the theorem.

At this stage we can also fully fill the environment directive called *requirements*. Most of the times this consists of all 5 articles[5], as mentioned in Section 3.3.1, whose content is known automatically by the Mizar system and allows some automatisa-

[4] `constr -f SQUARE_1:def 4`

[5] Five articles: BOOLE, SUBSET, NUMERALS, REAL, ARITHM are used by the Mizar system to provide automatic arithmetic operations, see [NB04] for more details.

tion of arithmetic operations[NB02, NB04]. On the other hand it is a recognised practice (in the Mizar community) to place all of those articles in this directive at the very beginning of the building of a Mizar environment for an article.

When we run the Mizar system now, it still responds with 2 errors. It also stops at the parser level of the Mizar system checker while validating the statement[6] (see Figure 5.13).

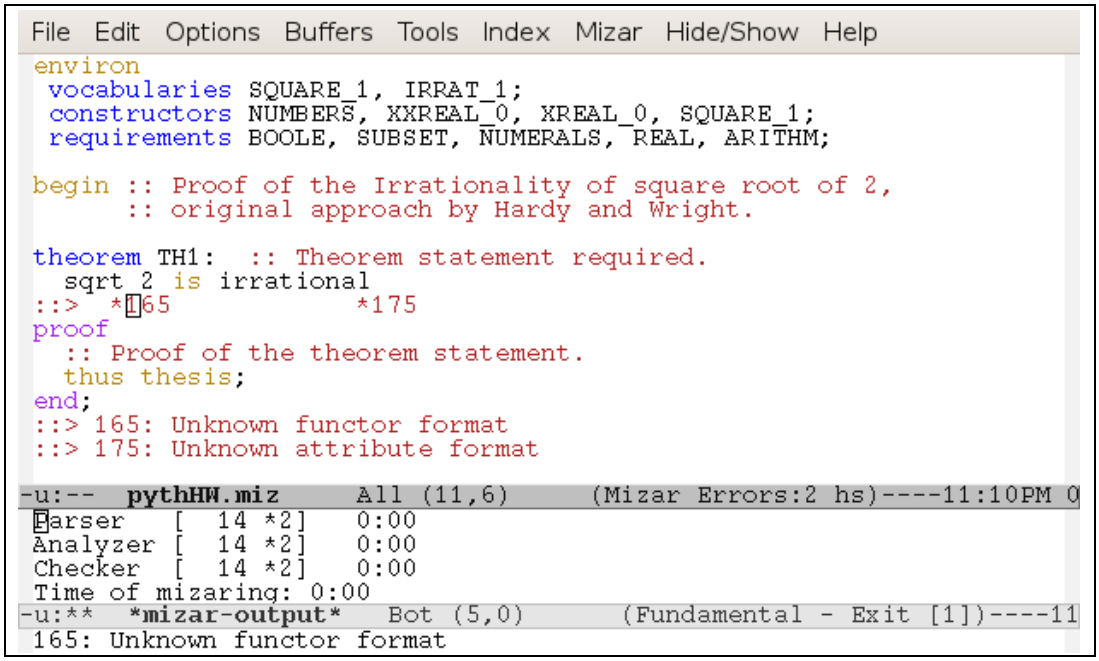


Figure 5.13: A part of the Mizar article environment for the theorem statement of Figure 5.2's example. The Mizar system responds with 2 errors.

We need to eliminate these errors and build an environment which would allow the Mizar system to check the sentence. Furthermore, the Mizar checker should provide only one "error" feedback stating that the theorem requires justification.

The adjective (Mizar *attr*) "irrational" has been introduced as an `antonym` for the original definition of the adjective *rational* (see the adjective definition `RAT_1:attr 1` in article `RAT_1`). Therefore, we need to place in the "vocabularies" directive the name of the article `RAT_1`. We also need to place some articles names in the directive *notations*, which are responsible to recognise the syntax of the

[6]The Mizar checker, while verifying a Mizar article, processes it in three stages: (1) Parser, (2) Analyzer and (3) Checker. The first one, as the name suggests, is a parser which reads the article and parses each statement trying to identify the format of each element of the statement according to vocabulary and notations. The analyzer, checks the meaning of each element and analyses if the identifier is properly used. Moreover it transforms the statements into Mizar internal formats which are used later by the checker.

symbols and the identifiers used in the Mizar article. Hence we placed in this directive 3 articles: SQUARE_1, RAT_1 and IRRAT_1.

It is important to note that the adjective "irrational" has been introduced for any "real number" (see the notation introduced in article IRRAT_1 (i.e., IRRAT_1:attr 1)). Therefore, first we need to identify the name "number" and "real", and to find the article in which these names have been introduced, i.e., ARYTM.

Now we have to identify where the definition of the adjective "real" has been introduced. The article name is XREAL_0, and we need to place it within the *notations* directive.

At this stage, the Mizar system still does not recognise automatically a type: "real number". We need to identify an article in which the registration (a theorem which can provide an introduction to a new type – a cluster of an adjective and a radix type: mode or structure) has been introduced, so the Mizar system can automatically identify a type: "real number". This has been introduced within the article: XREAL_0.

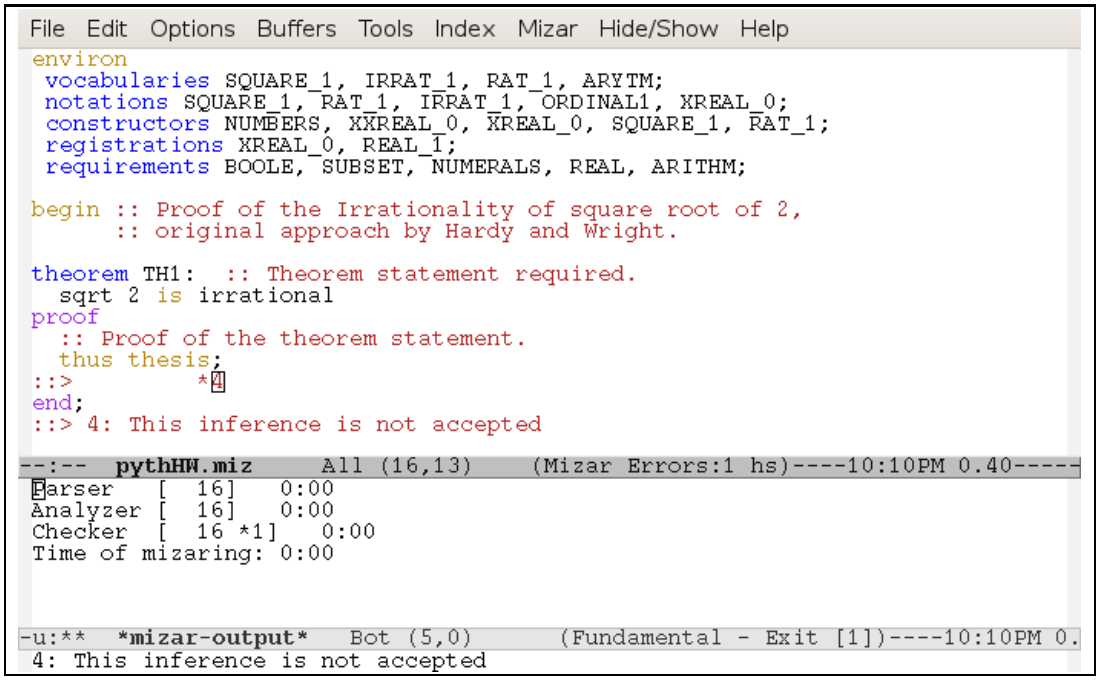


Figure 5.14: The Mizar FPS environment built for the example from Figure 5.2. The Mizar system responds with only one error stating that the theorem is not sufficiently proved.

Within the next step we want to find out that "sqrt 2" is a real number. This can be found in the article REAL_1. Finally with such a built environment (see Figure 5.14) the Mizar system responds with only one error, the error no. *4

stating that "the inference is not accepted", which basically means that it is not sufficiently proved or that a proof of the statement is missing. In our case we need to provide a proof for the statement.

At this stage the user has to translate the proof of the theorem into the Mizar language. While transforming each statement from the MathLang presentation into the Mizar language the user needs to identify and fullfill parts of the environment in a similar way to that presented above. The environment has to be filled to the stage, where the Mizar system responds with only two kinds of errors: **4* and **1*. These two errors indicate inferences and proofs of parts of the document that are not sufficient and that still contain reasoning holes, therefore the Mizar system can not fully verify the document. The full content of the Mizar FPS version is presented in Figure 5.15.

```
File  Edit  Options  Buffers  Tools  Index  Mizar  Hide/Show  Help
environ
 vocabularies SQUARE_1, IRRAT_1, RAT_1, ARYTM,
   INT_1, ARYTM_3, MATRIX_2, ORDINAL2;
 notations SQUARE_1, RAT_1, IRRAT_1, ORDINAL1, XREAL_0,
   INT_1, INT_2, XCMPLX_0, ABIAN;
 constructors NUMBERS, XXREAL_0, XREAL_0, SQUARE_1, RAT_1,
   INT_2, XCMPLX_0, POWER, ABIAN;
 registrations XREAL_0, REAL_1,
   INT_1, NAT_1, SQUARE_1;
 requirements BOOLE, SUBSET, NUMERALS, REAL, ARITHM;

begin :: Proof of the Irrationality of square root of 2,
      :: original approach by Hardy and Wright.

theorem TH1:  :: Theorem statement required.
  sqrt 2 is irrational
proof
  :: Proof of the theorem statement.
  assume sqrt 2 is rational;
  consider a,b being Integer such that
  a^2 = 2 * b^2 and
  a,b are_relative_prime;
::>                    *4
  a^2 is even;
::>          *4
  a is even;
::>       *4
  consider c being natural number such that
  a = 2*c;
::>     *4
  4*c^2 = 2*b^2;
::>           *4
  2*c^2 = b^2;
::>          *4
  b is even;
::>       *4
  thus contradiction;
::>                *1
end;
::> 1: It is not true
::> 4: This inference is not accepted
--:--  pythHW.miz     All (38,20)    (Mizar Errors:8 hs)----12:30AM 0.40----------
Parser   [  31]   0:00
Analyzer [  31]   0:00
Checker  [  31 *8]   0:00
Time of mizaring: 0:00


-u:**  *mizar-output*   Bot (5,0)      (Fundamental - Exit [1])----12:30AM 0.40---
1: It is not true
```

Figure 5.15: The Mizar Formal Proof Sketch of our example shown in Figure 5.2.

At this point we achieve the Mizar FPS version of the original document (see Figure 5.2). We can equally compare the view of the original document with the Mizar FPS version, as well as the MathLang version of the original document. The next section presents a description of a path from Mizar FPS to full Mizar.

5.2.4 The Mizar FPS version of the CML to full Mizar

Previous section shows how a user transforms the MathLang computerised version of the original document into the the Mizar FPS version. To complete our proposed gradual path of formalisation mathematical documents, the user needs to fill out missing parts of the proof in the Mizar FPS version of the document. The user needs to fill the proof up to the level that the Mizar system accepts a document without errors.

To do this, the user needs to possess more expert knowledge of Mizar and the MML than it was required while building the Mizar FPS. The essential knowledge is the ability to find definitions, theorems and clusters within the MML that are mandatory to full fill some justification holes within the proof. At this level, the user also needs to fill in appropriate directives in the environment, which is not an easy task, especially filling in the "notations" directive, where the order of articles can make a big difference for the Mizar system/checker.

Figure 5.16 presents the Mizar FPS proof (left hand side of the Figure) together with the full formalised version of the proof (right hand side of the Figure), where lines of the Mizar FPS proof version match lines of the full proof version.

There are number of important issues that has to be discussed of this full version of the Mizar FPS proof.

Firstly, the Mizar FPS version shows places where the Mizar system labelled its errors, mainly justification holes. On the Mizar full formalisation of the proof, those holes have been filled out with justifications. Most of those justifications are actually "simple justifications" as called in the Mizar jargon or *Straightforward-Justification* as indicated in the Mizar syntax. In Mizar a "simple justification" (or *Straightforward-Justification*) is presented as a reference to the previously proved statement, either local or taken from the MML. The structure of "simple justification" is the keyword "by ..." followed by the label of the statement referred to.

Secondly, a reasoning logic of the CML version of the proof is intuitive and contains holes considered to be trivial and not worth mentioning from the human-being perspective. However, if we aim to provide a representation of the proof for

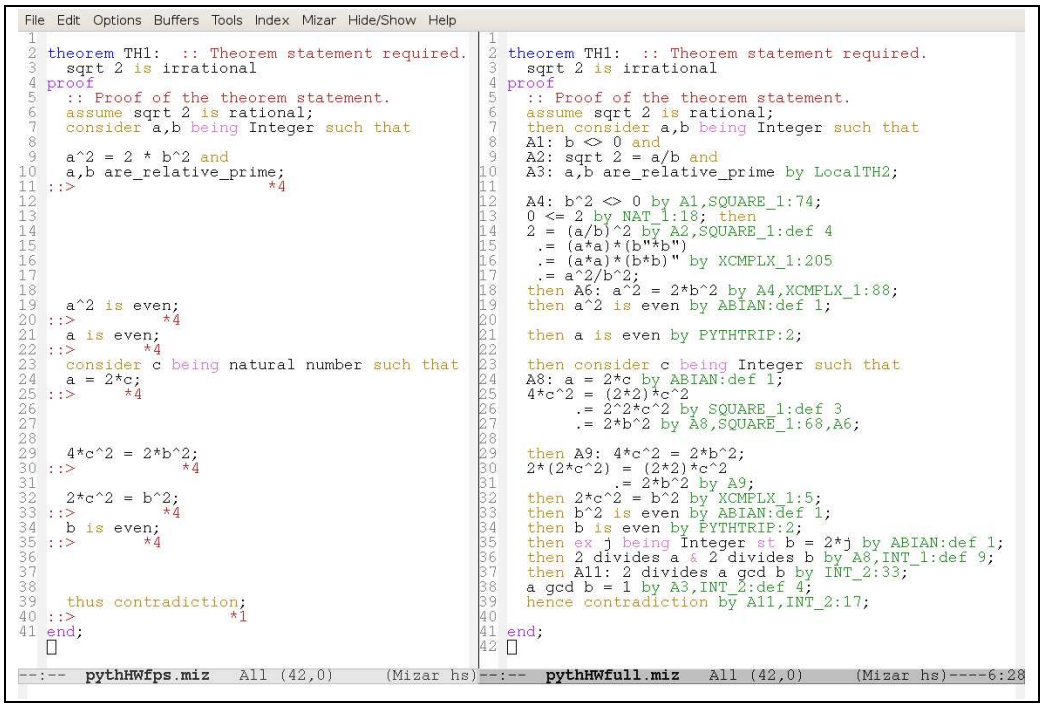


Figure 5.16: A comparison view of two representations of a proof of the Pythagoras' theorem shown in Figure 5.2. On the left hand side of the figure is the Mizar FPS proof, whereas on the right hand side is the full Mizar proof of the theorem.

a computer to verify its logic in the reasoning steps, we have to fill out all possible reasoning holes and provide a fully formalised version of the proof. Therefore some of the Mizar FPS holes requires additional statements being proved prior their usage in simple justifications. As a good example lets take a statement (1) `4*c^2 = 2*b^2`. Such statement required some arithmetic calculation and references to existing statements from the MML. This is due the fact that the Mizar is a proof assistant and proof checker, and is able to derive only some of reasoning steps (e.g., simple arithmetic, see article [NB04]) whereas most of them has to be made and written explicit by the user. As presented in the example, see right hand side of Figure 5.16, we provided additional 3 steps for the Mizar to be able to get it fully formalised and verified by the system.

Thirdly, we introduced two additional local theorems that are needed for this proof:

```
LocalTH1:
  for k,l being Integer st not k,l are_relative_prime holds
  ex x being Integer st x = k gcd l & x <> 1
```

and:

```
LocalTH2:
  for p being Rational holds
  ex a,b being Integer st b <> 0 & p = a/b & a,b are_relative_prime
```

The first local theorem `LocalTH1` is used inside a proof of the second local theorem `LocalTH2`. This theorem, `LocalTH1`, could actually be moved inside the proof of the theorem `LocalTH2`. However, for clarity of the proof we have placed this outside the proof.
The proof of the Pythagoras' theorem is a *reductio ad absurdum* (i.e., *proof by contradiction*). So it assumes that the thesis is false and that "$\sqrt{2}$ *is rational*". From such an assumption we conclude the information that is actually presented in theorem `LocalTH2`, which makes the theorem the base of the proof. However, this local theorem requires a proof itself. For this reason, we have placed it outside the original proof of the Pythagoras' theorem, see Listing A.3. It is also important to note, that the proof of the local theorem `LocalTH2` can have at least two different approaches, as presented on Listing A.3 and Listing A.4. The second Listing A.4 presents the proof proposed by A. Naumowicz during discussion of the author of the thesis and A. Naumowicz on the Mizar "developer-forum". This proof has been optimised to 3 lines! and uses different notations and definitions than those presented on Listing A.3.

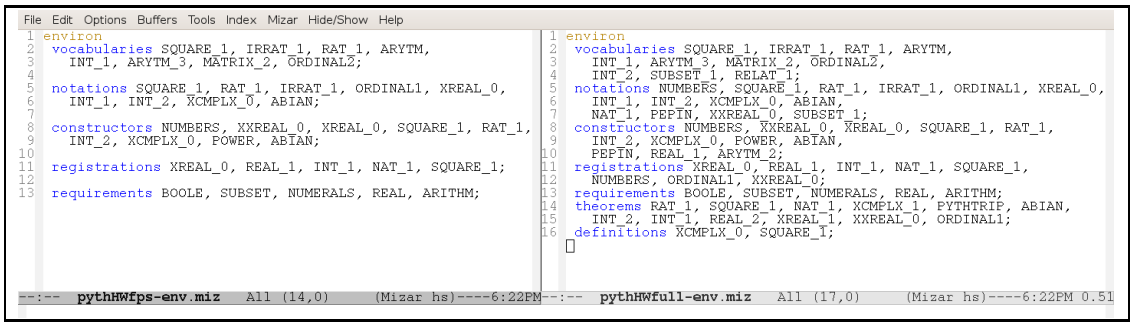


Figure 5.17: A comparison view of two representations of the environment for the Mizar representation of the Pythagoras' theorem shown in Figure 5.2. On the left hand side of the Figure is the Mizar FPS environment, whereas on the right hand side is the environment of the full formalisation of the proof of the theorem.

Fourthly, the environment for the full formalisation differs from the environment of the Mizar FPS version. Although, both contain the same parts (as shown on Figure 5.17 with the same line numbers and splits of the lines for both presentations), the fully formalised version is a bit more complicated. Moreover, the fully formalised environment contains a "theorems" directive and a "definitions" direc-

tive, which were not introduced in the Mizar FPS environment. In these directives the fully formalised version contains names of articles from which theorems and definitions where used to justify the reasoning of the proof.

At this level, it could sometimes appear that previously built Mizar FPS and choices of formalisation have to be reviewed, verified and changed to fit to the final goal and destination of the full formalisation. Moreover, as this simple example shows, in most of the cases, while transforming the Mizar FPS to the full formalisation, a user has to introduce additional definitions and theorems that are not yet introduced in MML. All that makes the transformation process from Mizar FPS to full Mizar very labor intensive and time consuming, especially for unexperienced Mizar users. Therefore, we recommend that at this stage a Mizar expert performs the transformation and full formalisation.

5.3 Narrative Features vs. Mizar Text-Proper Skeletons

The purpose of the DRa is to discern explicitly the structure of mathematical knowledge for providing a better encoding of its content, see Figure 4.6. 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 4.5, the DRa annotations of a text are used to automatically generate the dependency graph of the text where the relationships between different parts of a 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 5.20). 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 makes explicit 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.

As the main example for presenting transformation hints from the DRa annotated document to the skeleton of the Mizar FPS (see Section 5.3.1), we use the proof of Pythagoras' theorem by H. Barendregt as shown in Figure 4.6. As an illustration of a dependency graph we refer to two examples: the H. Barendregt example, and the example by J.M. Möller shown in Figure A.1. We also provide a short

description of deriving a Mizar FPS skeleton from the DRa annotation for both of these examples.

5.3.1 Transformation Hints Provided by the Dependency Graph

Note that the DRa dependency graph (see the left hand side of Figure 4.11) does not impose any logical correctness. For instance, on the example of H. Barendregt proof of the Pythagoras' theorem (see Figure 4.6), the paragraph labeled proof is related by the relationship justifies to a mathematical sentence labeled corollary (see Figure 4.8). 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 DRa. We expect to get an automatic validation of the coherence of the relationships drawn in the document (for instance a block of steps labeled proof can not "justifies" another step labeled axiom).

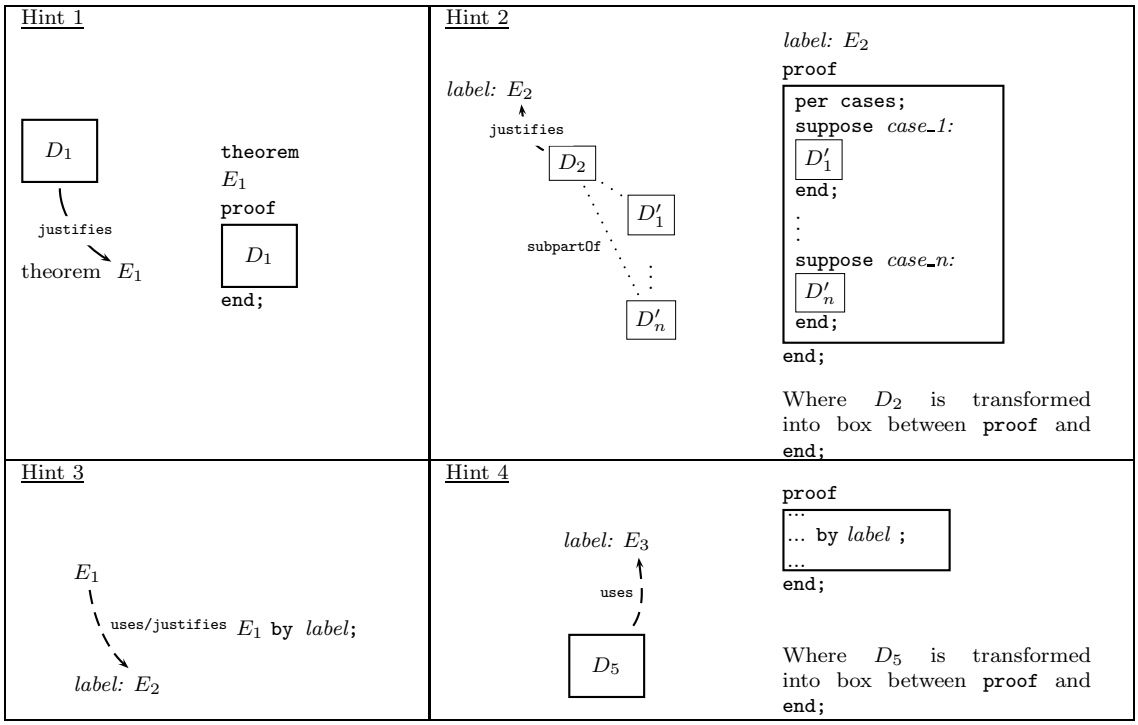


Figure 5.18: Four transformation hints provided by the dependency graph.

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.

Figure 5.18 lists four hints that we use to transform the example of H. Barendregt proof of the Pythagoras' theorem. 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 us 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 related via justifies to 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 the user a hint that this is a *Proof Justification* in which the 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* and has to be presented in a form of a keyword "by ..." followed by the label of the statement to which the user refers to.

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.

5.3.2 From the Document Narrative Structure to the Formal Document Skeleton in Formal Systems

A text annotated by a mathematician with the DRa aspect has been used to automatically produce the Dependency Graph (DG) and the Graph of Logical Precedences (GoLP) of the text. Such annotated DRa aspect and produced graphs make explicit the narrative, structural and logical features of the text. In this section, we explain how the automatically generated dependency graph and GoLP are used for further processing and formalisation of the text. In particular, we express how the dependency graph together with the GoLP are used to build a skeleton of a

part of a Mizar article – *Text-Proper*. We present this transformation for two examples: the first is presented in Section 5.3.2.1 for the example of H. Barendregt proof of Pythagoras' theorem and the second is presented in Section 5.3.2.2 for the J.M. Möller example (shown in Figure A.1).

5.3.2.1 The example of H. Barendregt proof of the Pythagoras' theorem – from the DRa to the Mizar FPS Text-Proper skeleton

Using the transformation hints of Figure 5.18, we can transform the dependency graph produced for the example of the proof of the Pythagoras' theorem by H. Barendregt (see the left hand side of Figure 5.19) into a proper structure of the *Text-Proper* part of our Mizar FPS (see the right hand side of Figure 5.19). 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.

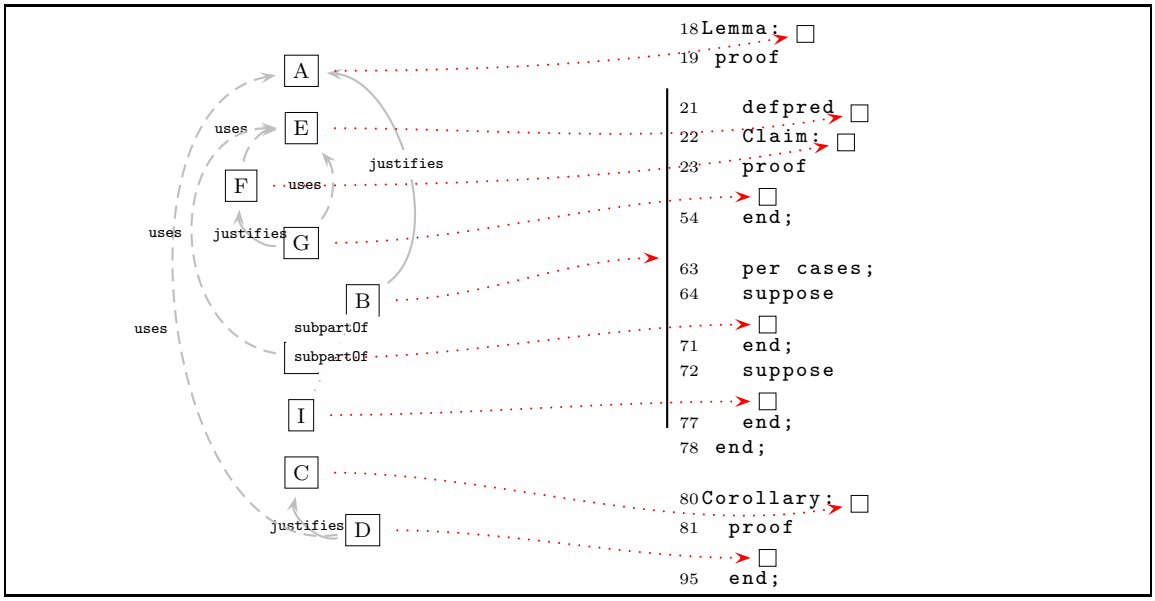


Figure 5.19: The transformation of the *dependency graph* of the proof of Figure 4.6 into Mizar *Text-Proper* skeleton.

The left hand side reproduces the MathLang dependency graph of the H. Barendregt approach to the proof of the Pythagoras' theorem (Figure 4.8). On the right hand side we show the Mizar *Text-Proper* skeleton of the same example. The arrows from left to right show how the MathLang dependency graph gives hints on how to build the Mizar *Text-Proper* skeleton.

5.3.2.2 The example of J.M. Möller – from the DRa annotation to the Mizar FPS the **Text-Proper** skeleton

In Section 4.3.1, the mathematician specified that a big box named $S2$ is an entire section in the document. In Mizar the *Text-Proper* part of a document could be divided into 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 Mizar document. Hence, the whole box is indicated to be a section by explicitly specifying `begin` at the very top of the right hand side of Figure 5.20. It also consists of two lines `::Section` and `::Title ...` which are treated as Mizar comments, and are solely oriented for the Mizar user consumption, or the reader of the Mizar file. Inside `::Title ...` it is a good practice (in the Mizar community) to specify the title of this *Section* of the Mizar document.

Since the mathematician specified for the box $\boxed{D1}$ the MathematicalRhetoricalRole definition, then it is transformed into Mizar syntax as: `definition :DEF1:` $\boxed{D1}$ `end;` (see Figure 5.20). In Mizar we introduce the label `DEF1` for this definition to be able to refer to it in further reasoning steps.

Since the mathematician specified for the box $\boxed{T1}$ the MathematicalRhetoricalRole theorem, then it is transformed into Mizar syntax as:
`theorem` $\boxed{T1}$. Moreover, since the box $\boxed{PT1}$ has the MathematicalRhetoricalRole proof, then we transform it into: `proof` $\boxed{1}$ `end;`. Moreover, since a block of steps having the mathematical role proof is related by justifies to a single statement, we can say that this is a *Justification* in Mizar, which is transformed into a specific form. See the corresponding transformation arrows in Figure 5.20.

In the dependency graph of our main example we also specified that some blocks of text use other blocks. For instance a block of text named $\boxed{PT2}$ uses statement $\boxed{L1}$. Here, we transform $\boxed{PT2}$ into a specific Mizar *Proof* block, which contains an expression with *Straightforward-Justification* to statement $\boxed{L1}$, where in Mizar it is reused by referring to a label (i.e., `LEM1`) that was assigned to a statement $\boxed{L1}$ during the transformation into the Mizar syntax.

During the transformation of the DG, we use the GoLP of our main example to be able to put annotated and named chunks of text into a proper Mizar order inside the Mizar skeleton.

The above transformation process leads to a part of a Mizar *Text-Proper* skeleton of a Mizar document (given in Figure 5.20 for our main example).

The grammatical information of the original text, which is captured by the

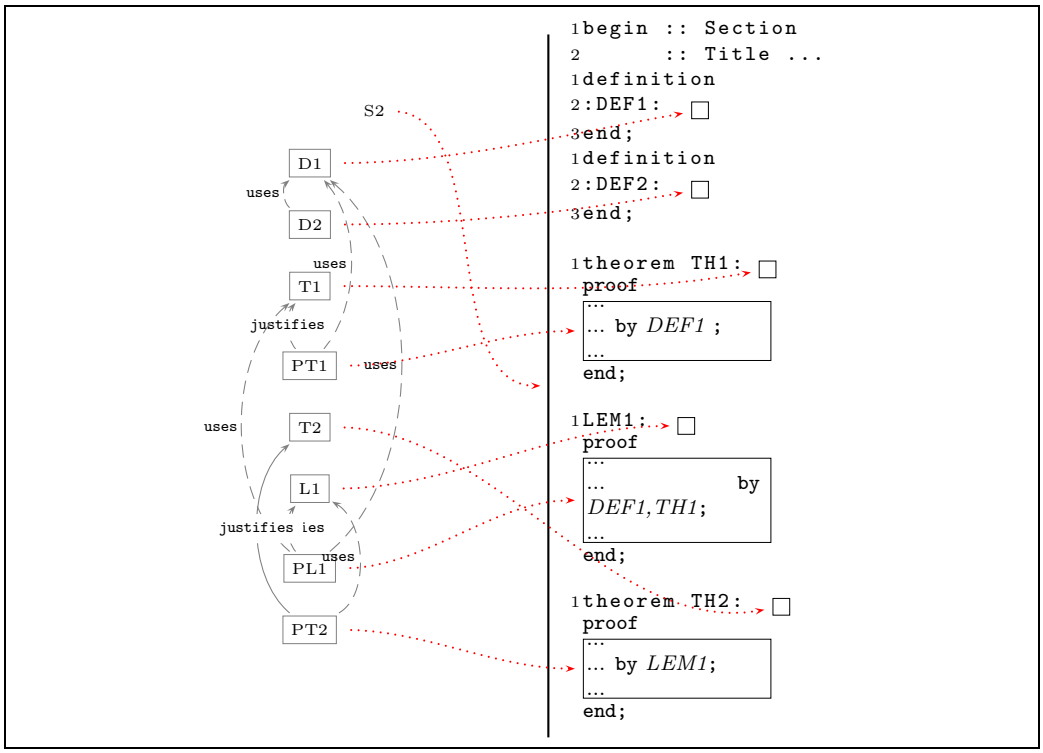


Figure 5.20: The transformation into Mizar skeleton.

The left hand side reproduces the dependency graph of our example (Figure A.1). On the right hand side we show the Mizar *Text-Proper* skeleton of the same example. The arrows from left to right show how the dependency graph is used to build the Mizar *Text-Proper* skeleton. □ stands for holes (incomplete proofs).

CGa aspect of MathLang and stored in the MathLang document, can then be used to fill more details in the current skeleton of the Mizar document. This better filled document could be transformed later into a proper Mizar document. The work describing these transformation and usage of the MathLang document for the migration process into the Mizar language, has been performed for the approach of H. Barendregt to the proof of Pythagoras' theorem and is described in the following section.

5.4 Building Parts of Mizar FPS from a Grammatically Annotated Document

This section describes how the annotation of the MathLang CGa aspect of a mathematical document can be used to build parts of the Mizar FPS translation of the document. These parts of a Mizar FPS include filling in some *directives* of an *envi-*

ronment, as well as translating some of the MathLang CML annotated statements into the Mizar language. We present here results of the research, that we believe will provide in a near future, a semi-automated way of building Mizar FPS from the MathLang annotated document. We explain these results by covering them with the example of the H. Barendregt approach to the proof of the Pythagoras' theorem. In this section we will refer mainly to this example, rarely providing a fully qualified name (i.e., the example of H. Barendregt proof of Pythagoras' theorem).

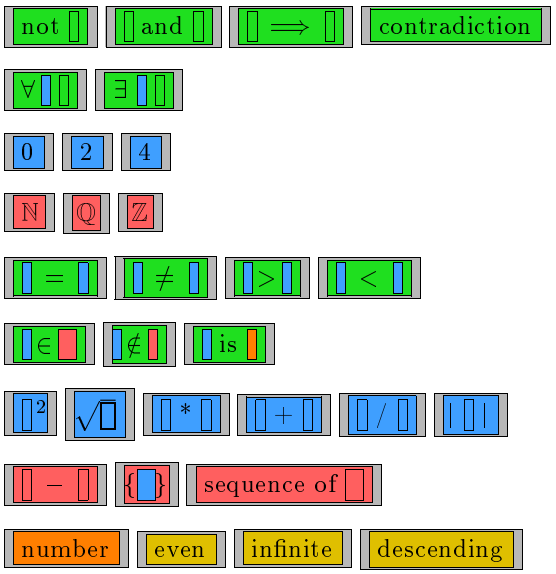
5.4.1 The document's background knowledge

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 forms an important part of the background knowledge required to understand it. For example, the arithmetic operations plus ($+$), times ($*$) or square root ($\surd$) are not defined in our main example (see Figure 4.6) 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 5.21). At the DRa level, we identified this list of declarations as a **preamble** by annotating the CGa paragraph containing the left hand side of Figure 5.21 by:

[description hasStructuralRhetoricalRole="preamble"]

In our example, the identifier even (line 28 of Figure 5.21) 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.

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 5.2 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




```
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; };
```

Figure 5.21: The **preamble** of MathLang's encoding of Figure 4.6 example. 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 [KMW06]).

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 5.2: the Mizar FPS *Environment*

```
6environ
7 vocabularies INT_1, SQUARE_1, MATRIX_2, IRRAT_1, RAT_1, ARYTM_3, ABSVALUE,
8   SEQM_3, FINSET_1;
9 notations INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1, XCMPLX_0,
10   INT_2, SEQM_3, FINSET_1, REAL_1, PEPIN;
11 constructors INT_1, NAT_1, SQUARE_1, XXREAL_0, ABIAN, RAT_1, IRRAT_1,XCMPLX_0,
12    INT_2, SEQM_3, FINSET_1, PEPIN;
13 requirements SUBSET, NUMERALS, ARITHM, BOOLE, REAL;
14 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 these *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 here 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.

5.4.2 Mathematical identifiers and their formal counterparts

5.4.2.1 Mathematical conjuctions.

In the **preamble** created in our example (see Figure 5.21) one can find the introduced identifiers that play the role of statement conjunctions, e.g. *implies* or *and*. These are usually reserved words and terminals 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	*Formula* `implies` *Formula*
iff	*id* is choosen by the user, e.g.	*Formula* `iff` *Formula*
not	lines 4–7 in Figure 5.21.	`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.

5.4.2.2 Binders

Binders like '∀', '∃' or '$\sum$' are indispensable parts of CML. In CGa, the user has to declare them (see figure 5.21). 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 [Wiea]. Nonetheless, the Mizar language offers the two most essential binders: ∀ and ∃ which are given as *Quantified-Formula* in the Mizar syntax (see the table below).

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*)

5.4.2.3 Functions

Function 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 returns a value with the same type as argument (see the table below).

CML	_ ^ 2
Possible CGa	19 sq(term): term;
Mizar	definition let x be complex number; func x^2 equals :: SQUARE_1:def 3 ... end;

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	_ \ _
Possible CGa	25 subtraction(set,set): set;
Mizar	definition let X,Y be set; func X \ Y -> set means :: XBOOLE_0:def 4 ... end;

5.4.2.4 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;

When we use a `synonym` or `antonym`, Mizar will internally use the real name. So if we write `a < b`, then Mizar will consider this as an abbreviation of `not b <= a`.

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;

5.4.2.5 Nouns

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 below). 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 [KMW06]) which 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`

5.4.2.6 Adjectives

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* [Ban03].

CML	*even*
Possible CGa	`28 even: adj;`

Mizar	`definition let i be number;`
	`  attr i is even means`
	`:: ABIAN:def 1`
	`  ...`
	`end;`

5.4.2.7 Other identifiers

In CGa we have declared some identifiers to be terms, e.g. 0, 2, 4 (see line 10 of Figure 5.21), 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 5.21). 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.

5.4.3 Transforming the document building steps

As already mentioned (see Section 2.3), 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.

5.4.3.1 Atomic statements

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 ;`

5.4.3.2 Blocks

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 5.3).

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`

5.4.3.3 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.

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;

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 does not 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	So $m = 2k$ and we have
Possible CGa	46 { k: N; =(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 corresponding Mizar structure.

5.5 Conclusion

In this chapter we described the idea of gradual computerisation of mathematical documents starting from the CML version of a mathematical document and finishing with its full formalisation in Mizar. We explained how this computerisation process is done in a number of gradual steps following MathLang's approach and its aspects.

In Section 5.1 we described different possible formalisation paths which are also shown in Figure 5.1. In the same section we have also briefly explained the following paths:

1. The direct path from CML to Mizar.
2. The path from CML to Mizar via Mizar FPS as the middle stage.
3. The path from CML to Mizar following MathLang aspects and Mizar FPS.

In Section 5.2, we described in more details the computerisation path following MathLang's approach. We based this description on an example of the proof of the Pythagoras' theorem by G.H. Hardy and E.M. Wright (see Figure 5.2). First we showed the annotation of the CML version with the MathLang Core Grammatical aspect (CGa) and Text & Symbol aspect (TSa). Then we refined this annotation with our proposed Document Rhetorical aspect (DRa). Finally we presented the transformation of the MathLang computerised document into the correct Mizar FPS version. We also briefly showed the transformation of the Mizar FPS version into full Mizar.

Thereafter, we described in Section 5.3 how the DRa aspect provides the knowledge required to build the skeleton of a Mizar article. At this stage, we expressed a number of hints that allowed the transformation. Moreover, we believe that these hints can be used as a starting point for building a computer software which assists the user during the transformation of the DRa annotated document into the skeleton of the Mizar FPS version. We presented this transformation for two different examples: firstly for the proof of the Pythagoras' theorem by H. Barendregt, and secondly for the J.M. Möller parts of a text regarding topology.

In the next section we showed how the CGa + TSa annotation helps to build parts of a Mizar document. We have also provided a short comparison of different MathLang patterns and their counterparts in the Mizar language constructs (e.g., MathLang's "adjective" is an "attribute" in Mizar). This provides a generic knowledge for filling some of the directives of the environment in a Mizar article. Such knowledge also offers hints for converting MathLang annotated statements into proper formulas in the Mizar language. This is of course not an easy task, however, we believe that the use of the MathLang CGa and TSa annotation provides knowledge and hints which make building versions in Mizar FPS much easier.

Chapter 6

Implementations

> Language designers want to design the perfect language. They want to be able to say, "My language is perfect. It can do everything." But it's just plain impossible to design a perfect language, because there are two ways to look at a language. One way is by looking at what can be done with that language. The other is by looking at how we feel using that language – how we feel while programming.
>
> Because of the Turing completeness theory, everything one Turing-complete language can do can theoretically be done by another Turing-complete language, but at a different cost. You can do everything in assembler, but no one wants to program in assembler anymore. From the viewpoint of what you can do, therefore, languages do differ – but the differences are limited. [...]
>
> *Yukihiro Matsumoto*[1]

The above fragment expresses thoughts that need to be addressed while designing and implementing any type of computer or formal language. It should particularly concern developing formal languages for computerising and formalising mathematics.

Formal languages and frameworks are more difficult to develop and implement than any computer language. The reason for that is, that almost any newly developed computer language can be interesting enough so a significant number of people will try to use and test the new language. With formal languages it is the reverse. Hardly any mathematician or scientist uses formal languages for computerising and checking the correctness of his mathematical theories. This is due to

[1] Also known as "Matz". The author, designer and main developer of the Ruby programming language – `http://www.ruby-lang.org/en/`

various reasons. Due to these reasons only a small number of mathematicians express interests in using formal languages[2]. This leads to a slower development of the language itself, of the testing of the language in terms of the formalisation of real mathematics using that language, of bug recovery and fixing, etc.

Therefore it is important to address a main objective during the development and implementation of formal languages for computerising mathematics. The objective can be stated in the form of the question "how do we feel while computerising or formalising mathematics"?

In this chapter we focus on the implementation side of the MathLang DRa aspect. Furthermore, we show how this implementation addresses the question stated above. In Section 6.1 we provide a description of the concrete syntax for the DRa. We present both an XML syntax as well as a "plain" syntax, where both are internal presentations of the MathLang document and are not meant to be read by the MathLang user. In Section 6.2 we discuss our choice for the graphical user interface (GUI) for the MathLang framework. We provide description of the implementation of the DRa aspect as a MathLang plugin for the GUI. In Section 6.3 we describe the implementation of algorithms for producing graph representations of the DRa annotated documents. We present the implementation on the GUI side as well as the implementation of the XSL stylesheets used on a pure XML MathLang document. Finally in Section 6.4 we present the informal algorithm for checking the well-formation and annotation of the MathLang DRa aspect.

6.1 DRa Concrete Syntax

Since the start of the MathLang project the focus has always been to use existing techniques and standards in the implementation and development of the framework. Therefore, we use the XML recommendation of the World Wide Web Consortium (W3C[4]) to represent the MathLang document. The use of the XML standard makes the actual encoding of a mathematical document more accessible for computers, than any other text based presentation of the document. More importantly the XML presentation allows further manipulation or transformation of the encoded document. It is important to note that the internal XML presentation of the MathLang document is not meant to be read by a human and is purely oriented

[2]For instance, the Association of Mizar Users (SUM[3]) has approximately 160 members, in which probably 10–25% are active users who formalise mathematics. Although, the Mizar group is quite small, the development of the Mizar system is quite active.

[4]`http://www.w3.org/XML/`

for computers.

The XML concrete syntax and scheme have been developed for the first two aspects of MathLang, namely the CGa and the TSa. The development has been carried on by M. Maarek as a result of number of discussions within the MathLang group. M. Maarek has implemented those two aspects and provided an extensive description of the CGa and TSa XML syntax in his PhD thesis [Maa07].

The XML scheme and concrete syntax for the DRa aspect have been developed by the author of this thesis as an outcome of a number of discussions within the MathLang group. The *plain syntax* and the *concrete syntax* have been discussed in Section 4.4. In this section we present the summary and the XML scheme of the DRa.

6.1.1 Document Rhetorical namespace

Namespace URI (usual prefix: `dra`)
`http://www.macs.hw.ac.uk/ultra/mathlang/document-rhetorical`

This namespace contains elements and attributes that can be used to encode the DRa entities within the MathLang encoded document. The DRa entities have been introduced and discussed in Section 4.2. Table 6.1 provides a short comparison between the *plain syntax* and the *concrete XML syntax* of DRa.

6.1.2 The XML scheme of DRa

The DRa grammar is part of the XML scheme for the MathLang framework. The MathLang grammar has been written using the RELAX NG[5] compact syntax. We developed the RELAX NG scheme for the MathLang DRa aspect. Listing 6.1 presents a RELAX NG scheme of the XML syntax (i.e., XML elements and their XML children) for the MathLang DRa annotation.

We use the prefixes `mathlang`, `cga`, `dra`, `dc` for XML elements to indicate the namespace in which a particular element lives. The `dc` prefix refers to Dublin Core[6] (DC) and is used to encode metadata information. Generally speaking, the DC is a standardised way for describing a wide range of network resources. In the case of the MathLang framework we use DC to provide metadata description for a whole MathLang document or a particular element of the document, for instance

[5]REgular LAnguage for XML Next Generation is a simple schema language for XML. A RELAX NG scheme specifies a pattern for the structure and content of the XML document. RELAX NG scheme is itself an XML document but also contains compact non XML syntax.

[6]`http://dublincore.org`

	Plain Syntax
1 2 3	[node ddid@*node-id cga:step*] [about@*node-id* role-attrib@*node-role*]; [did@*relation-id* src@*source-node* type@*relation-type* anc@*anchor-node*]; where role-attrib can have values: hasMRR, hasSRR, hasOMRR, hasOSRR
	Concrete Syntax – XML
1	<dra:node xml:id="..."> <cga:step ...>...</cga:step> </dra:node>
2	<dra:description about="..." role-attrib="..." /> where role-attrib can have values: hasMathematicalRhetoricalRole, hasStructuralRhetoricalRole, hasOtherMathematicalRhetoricalRole, hasOtherStructuralRhetoricalRole
3	<dra:relation xml:id="..." src="..." type="relation-type" anc="..."/> where relation-type can have values: relatesTo, justifies, subpartOf, uses, exemplifies, inconsistentWith

Table 6.1: The *plain syntax* and the *concrete syntax* for MathLang-DRa
A short comparison between the *plain syntax* (the top part of the table) and the XML *concrete syntax* (the bottom part of the table). Each position of the top part of the table corresponds to the position of the bottom part.

CGa steps or DRa annotations. The DC is used to provide information like: title, author, creator, publisher, creation date and many more of the annotated resource.

In the scheme presented in Listing 6.1, we refer to the CGa step. The scheme for the CGa XML syntax has been extensively described in M. Maarek PhD thesis.

```
# DRa - Document Rhetorical aspect - RELAX NG scheme
# (using RELAX NG compact syntax)

namespace mathlang="http://www.macs.hw.ac.uk/ultra/mathlang"
namespace
cga="http://www.macs.hw.ac.uk/ultra/mathlang/grammatical-core"
namespace
dra="http://www.macs.hw.ac.uk/ultra/mathlang/document-rhetorical"
namespace dc="http://purl.org/dc/elements/1.1"

document = element mathlang:mathlang {
        metadata.information+ & cga.step+ }
```

```
cga.step = cga.phrase* & cga.local-scoping* & cga.step* &
         cga.others* & dra.annotation* & metadata.information*

dra.annotation =  dra.node* & dra.relation* & dra.description*

dra.node = element dra:node {
        xmlid.attrib, metadata.information,
        (cga.step+ | dra.relation* | dra.description*)}

dra.relation = element dra:relation {
        xmlid.attrib, attribute src { text },
        attribute type { dra.rel.types },attribute anc { text } }

dra.description = element dra:description {
                attribute about { text },
                ( dra.mrr.attrib | dra.srr.attrib
                  | dra.omrr.attrib | dra.osrr.attrib ) }

dra.rel.types = token "relatesTo" | token "justifies"
        | token "subpartOf"     | token "uses" | token "exemplifies"
        | token "inconsistentWith"

dra.mrr.attrib = attribute hasMathematicalRhetoricalRole {
        token "theorem" | token "lemma" | token "definition"
        | token "proof" | token "corollary" | token "axiom"
        | token "conjecture" | token "proposition"
        | token "example" | token "assertion" }

dra.srr.attrib = attribute hasStructuralRhetoricalRole {
        token "preamble" | token "part" | token "chapter"
        | token "section" | token "subsection" | token "paragraph" }

dra.omrr.attrib =
        attribute hasOtherMathematicalRhetoricalRole { text }

dra.osrr.attrib =
        attribute hasOtherStructuralRhetoricalRole { text }


# Metadata information is expressed as elements of
# the Dublin Core metadata standard.
metadata.information = element dc:* { ... }
```

```
# A unique id in the whole XML document
xmlid.attrib = attribute xml:id {text}
```

Listing 6.1: The RELAX NG scheme for the DRa grammar. The symbol "—" denotes the alternative, "*" the zero or more occurrence, "+" the one or more occurrence, "&" interleaving, where child elements are allowed in any order. The "token" is a built in datatype of RELAX NG and corresponds to the default behaviour of the literal string pattern.

6.2 TEX$_{\text{macs}}$ Side Implementation

This section summarises the implementation we developed to handle the annotation of the narrative structure of mathematical documents and to provide a user interface prototype.

6.2.1 The DRa editing tool

One of the challenges while developing any formal language is the development of the user interface/editor. The majority of existing formal languages, used for formalising mathematics, allow to use any text editor to write and/or edit documents formalised in that language. However, the use of the text editor is far from the mathematicians way of writing mathematics using computers. Even though LaTeX is mainly used by mathematicians to communicate or present their results, we believe that mathematicians still try to avoid formal languages due to their complexity and to being labour intensive. We believe that one of the reasons is the interaction with the mathematical document through the user interface (i.e., text editor), which at the moment is similar to computer programming/coding rather then expressing mathematical facts.

Therefore, from the very beginning of the MathLang project, the focus was on integrating the MathLang framework within a user friendly Integrated Development Environment (IDE). When we look purely at programming languages, we can notice an increasing number of IDE's available online. Such IDE's provides comprehensive facilities to programmers for software development, and more importantly they consist of a source code editor, a compiler/interpreter, a debugger and automation tools. All that makes an IDE a perfect tool for the programmer, which in result it makes the development process more rapid.

We believe that at present there is no tool that provides IDE's facilities for the mathematician who wants to computerise and further formalise mathematics. However, there exists at least one editor which has a great future of integrating the mentioned IDE's facilities within one tool for mathematicians. The TEXMACS editor is a What You See Is What You Get word processor, that allows the user to customise the editor to partly act as a manual typesetting program like TEX. Furthermore, TEXMACS is already used as an interface for computer algebra systemCAS (e.g., Axiom[7], Maxima[8], Sage[9]), numeral analysis, statistic systems, etc.

The first MathLang aspects, CGa and TSa, were implemented as *plugin* for TEXMACS. The plugin has been extensively described in M. Maarek PhD thesis. We extended the plugin with the DRa annotation facilities.

The DRa part of the MathLang plugin provides several ways to input DRa elements in a TEXMACS document. The user can access DRa functions via a menu and via keyboard shortcuts.

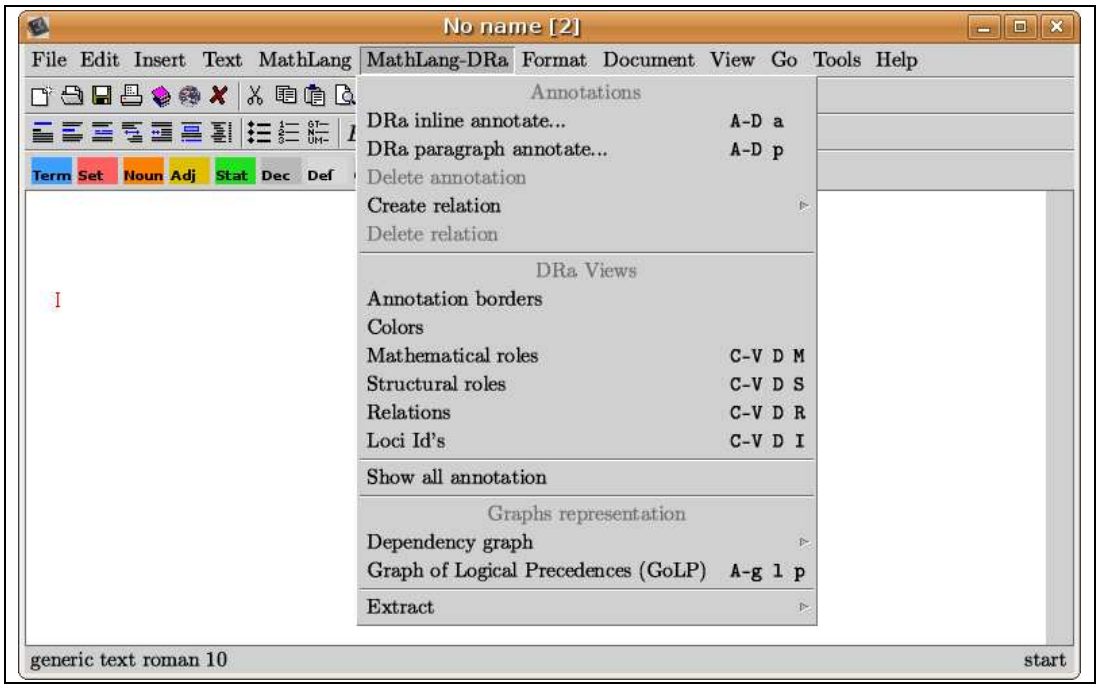


Figure 6.1: The MathLang plugin DRa menu in TEXMACS.

Similarly to the description in M. Maarek thesis (pp. 171), the menu (see Figure 6.1 of the DRa part of the MathLang plugin is divided into three groups:

[7] http://www.axiom-developer.org/
[8] http://maxima.sourceforge.net/
[9] http://www.sagemath.org/

1. The *Annotations* group which provides the facility for annotating DRa elements either inline or paragraph. It also provides the ability to create relations and delete an annotation or a relation.

2. The *DRa Views* group which gives the user the ability to produce/generate different views of a MathLang DRa annotation.

3. The *Graphs representation* group which allows the user to retrieve a DRa annotation from the annotated MathLang document, and to generate different graph presentations.

The DRa annotation process has been discussed in Section 5.2.2. In the same chapter we presented some views that can be generated from the MathLang encoded document, including the display of the DRa annotation.

6.2.2 DRa macros for TEXmacs

As described above, the mathematician who wants to annotate the Document Rhetorical aspect of a mathematical document, uses the DRa menu or his keyboard shortcuts for the MathLang plugin. Each use of the MathLang DRa function displayed in the menu, invokes specially developed TEXMACS macro which allows to annotate a DRa element on top of another TEXMACS or MathLang element.

For instance, let us recall the proposition from set theory which we presented in Figure 2.1. We annotate this mathematical statement with the MathLang DRa paragraph macro, see Figure 6.2

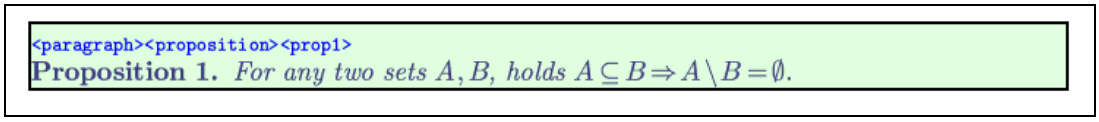


Figure 6.2: The MathLang DRa annotation for a set theory proposition.

The TEXMACS internal presentation of the document is stored in a tree-like format. This could be printed as a tree source for the above encoding (using the TEXMACS built in function chosen from the TEXMACS menu). The actual MathLang DRa presentation of such annotation is shown in Figure 6.3.

This example illustrates the TEXMACS element macro with 5 arguments which is used for annotating DRa entities around chunk of texts, see Figure 6.4.

The first and second argument is the interpretation that the user inputs to state the structural and mathematical roles, respectively. The third argument is *loci-id*,

```
⟨MathLang-DRa-p|paragraph|proposition|prop1||
    ⟨proposition|
        For any two sets ⟨math | A, B⟩, holds ⟨math | A<subseteq>B <Rightarrow> A
        <backslash> B = <emptyset>.⟩⟩⟩
```

Figure 6.3: The TEXMACS source tree presentation of the example from Figure 6.2.

```
⟨assign|MathLang-DRa-p|⟨macro|struct-roles|math-roles|loci-id|feedback|body|⟨MathLang-DRa-gen-
eric-p|struct-roles|math-roles|loci-id|feedback|⟨locus|⟨id|loci-id⟩|body⟩⟩⟩⟩
```

Figure 6.4: The DRa macro used to annotate any paragraph with DRa entities.

which is the unique identifier[10] provided by the user for the annotated element. The fourth argument of the macro stores feedback returned by the MathLang DRa checker. This part is still under development, therefore it is not discussed in more details in this thesis. Finally, the last argument contains the text that we annotate.

Similarly, the annotation of the DRa relations uses TEXMACS macros. For instance, the example below displayed as a TEXMACS tree source for the annotation of the DRa justification relation between proof and theorem.

```
⟨MathLang-DRa-relation|justifies||PR1|TH1⟩
```

Figure 6.5: The TEXMACS source tree presentation of the DRa relation annotation.

To annotate the relation we use a different TEXMACS macro with 4 arguments, as shown in Figure 6.6.

The first argument is a relation type. The second is a feedback returned by the MathLang DRa checker, while validating the DRa annotation. The third and fourth arguments are the source and the target, respectively, of the relation.

All TEXMACS macros developed for the MathLang DRa annotation are stored in the TEXMACS style file `mathlang-dra.ts`. Such file is loaded when the MathLang plugin is initialised during opening TEXMACS editor. To give an impression of that file we present a screenshot, see Figure 6.7, of the beginning of that file, opened using TEXMACS editor.

[10]The identifier has to be unique in the entire document, across all MathLang annotated entities.

```
⟨assign|MathLang-DRa-relation|
  ⟨macro|link-type|feedback|link-source|link-target|
    ⟨locus|⟨link|link-type|⟨id|link-source⟩|⟨id|link-target⟩⟩|⟨MathLang-error-of-feedback|feed-
    back⟩⟨MathLang-DRa-show-func | MathLang-DRa-show-relations | <relation::(link-source,
    link-type, link-target)>⟩⟩⟩⟩
```

Figure 6.6: The DRa macro used to annotate relations.

```
Style:      MathLang-DRa-$Rev: 3238 $
Purpose:    Extension to MathLang plugin for TeXmacs with Document Rhetorical aspect (DRa)
            annotation system.
Copyright:  © 2007 by Krzysztof Retel
License:    Heriot-Watt University, ULTRA Group

⟨use-module|mathlang-dra⟩

Settings for displaying modes.

⟨assign|MathLang-DRa-show-identifiers|false⟩
⟨assign|MathLang-DRa-show-relations|false⟩
⟨assign|MathLang-DRa-show-mathematical-roles|false⟩
⟨assign|MathLang-DRa-show-structural-roles|false⟩
⟨assign|MathLang-DRa-show-colors|true⟩
⟨assign|MathLang-DRa-color|pastel green⟩
⟨assign|MathLang-DRa-border-width|0.7pt⟩
```

Figure 6.7: A fragment of the TEXMACS stylesheet file for the MathLang DRa plugin. It consists of macros developed to allow DRa annotations in the TEXMACS document.

6.2.3 The TEXmacs SCHEME implementation

The TEXMACS core system is developed in C++[11] and SCHEME[12] programming languages. However, the user interface and most of the editing functions of TEXMACS are written mainly in SCHEME[13]. The use of the SCHEME programming language allows potential developers to customise the behavior of TEXMACS and to write their own extensions to the editor.

In above sections 6.2.2 and 6.2.1, we described the mechanism for the use of annotating the DRa entities with the DRa menu. This mechanism and functions required the implementation in the SCHEME. We structured the implementation

[11] `http://www.research.att.com/~bs/C++.html`

[12] `http://groups.csail.mit.edu/mac/projects/scheme/`

[13] The SCHEME is a multi-paradigm programming language and is one of the main dialects of Lisp (`http://www.lisp.org/alu/home`). The other dialect of Lisp is Common Lisp (`http://en.wikipedia.org/wiki/Common_Lisp`)

into number of SCHEME files which are loaded while opening the TEXMACS editor with enabled MathLang plugin:

- `init-mathlang.scm` – This file was mainly developed by M. Maarek and is the initialisation file for the MathLang plugin. It loads all MathLang SCHEME files to start up the plugin within TEXMACS.

- `mathlang-dra.scm` – The main file responsible for loading all MathLang DRa SCHEME files. It also contains functions for displaying different views of DRa annotated document, like borders, colours, structural or mathematical roles, relations, identifiers. It also consists of TEXMACS and SCHEME definitions for the DRa annotation, i.e., inline, paragraph annotations of DRa narrative roles and DRa relations.

- `mathlang-dra-menu.scm` – As the name suggests, it consists of functions for binding MathLang DRa menu.

- `mathlang-dra-kbd.scm` – Defines functions and keyboard shortcuts for the DRa functions placed in the MathLang-DRa menu.

- `mathalng-dra-graphs.scm` – Contains functions needed to build graphs presentations of the DRa annotated document. Some of these functions are using external programs to draw and display graphs, i.e., `dot` and `gv` programs.

- `mathlang-dra-extract.scm` – This file contains definitions of functions that allow to extract `loci-id`'s of annotated chunks of text with DRa entities. These functions also allow to extract the DRa annotated relations and present them in a separate TEXMACS page, although this uses already build in TEXMACS functions.

Figure 6.8 presents an example of SCHEME implementation for the DRa keyboard shortcuts. The first part of this Figure defines wildcards prefixes and keyboard shortcuts. Second part map the wildcards prefixes to their explanations. The final part maps the wildcards prefixes to actual functions that are invoked once the user uses keyboard shortcuts, the prefix `A-` corresponds to the `Alt` key. For instance to annotate the DRa relation "justifies" we use the following keyboard shortcuts combination: `Alt+Shift+R j`.

```
File  Edit  Options  Buffers  Tools  Scheme  Help
;; Wildcards prefixes.
(kbd-wildcards pre
  ("dra-view" "mathlang-view D")
  ("dra-create-relation" "A-R")
  ("dra-delete-relation" "A-R d")
  ("dra-annotation" "A-D")
  ("dra-create-graph" "A-g")
)

;; Explain wildcards prefixes.
(kbd-map
  ("dra-view" "" "MathLang DRa views.")
  ("dra-create-relation" "" "Create MathLang-DRa relation.")
  ("dra-annotation" "" "MathLang rhetorical annotation.")
  ("dra-create-graph" "" "Creates MathLang-DRa graph.")
)

;; DRa commands.
(kbd-map  ;keyboard-extend-forever ;kbd-map
  (:mode in-mathlang-dra?)
  ("dra-view I" (toggle-mathlang-dra-show-identifiers))
  ("dra-view S" (toggle-mathlang-dra-show-structural-roles))
  ("dra-view M" (toggle-mathlang-dra-show-mathematical-roles))
  ("dra-view R" (toggle-mathlang-dra-show-relations))
  ("dra-annotation a" (interactive mathlang-dra-annotate-inline))
  ("dra-annotation p" (interactive mathlang-dra-annotate-paragraph))
  ("dra-annotation d a" (remove-in-place-ann))
  ("dra-create-relation j" (interactive mathlang-dra-annotate-relation-justifies))
  ("dra-create-relation u" (interactive mathlang-dra-annotate-relation-uses))
  ("dra-create-relation s" (interactive mathlang-dra-annotate-relation-subpartOf))
  ("dra-create-relation i" (interactive mathlang-dra-annotate-relation-inconsistentWith))
  ("dra-create-relation e" (interactive mathlang-dra-annotate-relation-exemplifies))
  ("dra-create-relation r" (interactive mathlang-dra-annotate-relation-relatesTo))
  ("dra-create-relation o" (interactive mathlang-dra-annotate-relation-other))
  ("dra-create-graph d s" (call-mathlang-dra2dg)) ;Dependency Graph (DG) -- simple
  ("dra-create-graph d c" (call-mathlang-dra2dg-all)) ;Dependency Graph (DG) -- comprehensive
  ("dra-create-graph l p" (call-mathlang-dra2golp)) ;Graph of Logical Precedences (GoLP)
  ("C-R" (mathlang-dra-reload))
  ("C-D" (call-delete-mathlang-dra-ann))
)
```

Figure 6.8: Fragments of the `mathlang-dra-kbd.scm` SCHEME file. It presents functions used to bind keyboard shortcuts to specific MathLang DRa annotation functions.

6.3 Transformation Tools for Annotated Document

One of the aims of MathLang framework is to allow further manipulations of the MathLang encoded document. This is easily possible due to the internal XML presentation of MathLang encoded documents. The use of the XML standards allows to apply XSL stylesheets to the MathLang document. This section presents the transformation files written to generate different views of a MathLang encoded document. We concentrate here on the development of stylesheet files for the MathLang DRa aspect that allow to transform the DRa annotation into graph presentation.

6.3.1 The XSL Transformations

XML documents can be transformed to any desired presentation and format by applying the appropriate stylesheet written in XSL[14] Transformations (XSLT) lan-

[14] "The Extensible Stylesheet Lanugage is a family of recommendations for defining XML document transformation and presentation." (according to `http://www.w3.org/Style/XSL/`).

guage. The XSLT is a language that allows to define transformation and presentation rules for an XML document. The XSLT uses the XML Path Language (XPath) – an expression language – to access or refer to parts of an XML document.

We developed a number of XSLT stylesheets to transform the MathLang DRa annotated document into different graph representations, like the Dependency Graph and the Graph of Logical Precedences. As an output of the XSLT transformations of the MathLang DRa document we get the description of a graph written in the DOT language[15]. Later on the produced `.dot` file[16], which stores the description of our graph, is passed to a computer program called `dot` to generate the Postscript file with a visual presentation of the graph. The `dot` computer program is part of the `Graphviz` layout package. The `Graphviz` takes descriptions of graphs in a text language and generates diagrams in several useful formats such as images, SVG, Postscript files etc.

Figure B.1 presents fragment of the XSLT stylesheet developed to transform MathLang DRa annotated relations into the description of the dependency graph in the DOT language.

All presented graphs in this thesis have been produced automatically by applying appropriate XSLT stylesheets, sometimes also written for the LaTeX annotated document. The next section will present how we use the DOT language and the transformation technology to generate graphs directly from the TeXmacs annotation.

6.3.2 The Scheme Implementation for Generating Graphs

The internal presentation of a TeXmacs document has an XML-like tree structure even though it uses different entities, see Listing 2.2. Furthermore, the TeXmacs Scheme implementation and functions allow to access each node of that document tree and transform it into another presentation. As a result, it provides a simple way to build a transformation file for the TeXmacs annotated document and finally for the MathLang encoded document in TeXmacs. However, the transformation functions and rules have to be written in Scheme and have to be integrated as an extension/plugin for TeXmacs.

We developed the transformation rules for the MathLang DRa encoded docu-

[15]The DOT language allows in a simple and easy way to describe graphs that both humans and computer can use. It has a very small abstract grammar `http://www.graphviz.org/doc/info/lang.html`

[16]The `.dot` is a file extension for DOT graphs.

ment, that allow to retrieve MathLang DRa annotation and build the description of DRa graphs in the DOT language. Moreover, such generated description is passed to the external computer program (called DOT) to compile and build the visual presentation of the graph in the form of a Postscript file. Once the Postscript file is generated it is passed to the `gv` program to display that file.

We integrated these transformation rules and functions within the `mathlang-dra-graphs.scm` file of the MathLang plugin. It actually follows a similar approach to the described above XSLT transformation files. Each time we call the function, from the MathLang-DRa menu to generate the graph presentation, it walks through the TEXMACS tree and builds an appropriate graph presentation in the DOT language. Figure B.2 presents the fragment of a function responsible for building the DG graph from the annotation. Figure 6.10 presents the automatically generated description of the graph for the Pythagoras Theorem example (see Figure 4.6) which is built from the DRa annotated MathLang document as presented in Figure A.6. Finally, Figure 6.9 presents two possible views of the DG graph of the document, generated automatically from the TEXMACS editor. The lefthand side of that Figure presents only id's and relations from the DRa document, whereas the righthand side of the Figure presents the full annotation information including structural and mathematical roles.

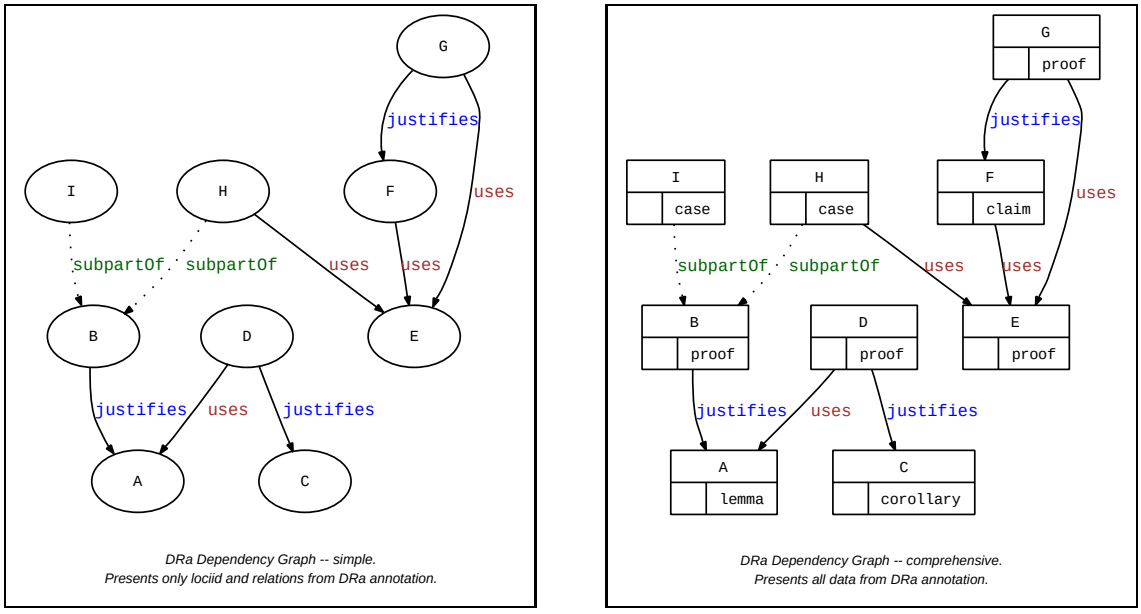


Figure 6.9: The presentation of the DRa Dependency Graph generated automatically from the TEXMACS, by the use of functions from the MathLang-DRa menu.

```
digraph mathlang_dra_dependency_graph_comprehensive{
 graph [ fontname = "Helvetica-Oblique",
         fontsize = 8,
         label = "\n\n DRa Dependency Graph -- comprehensive. \n
         Presents all data from DRa annotation. "
       ];

  node [ shape = record,
         fontsize = 9,
         fontname = "Courier"
       ];

  edge [ fontsize = 10,
         fontname = "Courier",
         arrowsize = .5
       ];

B -> A [style=solid, fontcolor=blue, label=justifies];
D -> C [style=solid, fontcolor=blue, label=justifies];
G -> F [style=solid, fontcolor=blue, label=justifies];
H -> B [style=dotted, fontcolor=darkgreen, label=subpartOf];
I -> B [style=dotted, fontcolor=darkgreen, label=subpartOf];
D -> A [style=solid, fontcolor=brown, label=uses];
H -> E [style=solid, fontcolor=brown, label=uses];
F -> E [style=solid, fontcolor=brown, label=uses];
G -> E [style=solid, fontcolor=brown, label=uses];
F [ label="{ F | {   |  claim } }" ];
H [ label="{ H | {   |  case } }" ];
I [ label="{ I | {   |  case } }" ];
G [ label="{ G | {   |  proof } }" ];
A [ label="{ A | {   |  lemma } }" ];
C [ label="{ C | {   |  corollary } }" ];
B [ label="{ B | {   |  proof } }" ];
E [ label="{ E | {   |  proof } }" ];
D [ label="{ D | {   |  proof } }" ];
}
```

Figure 6.10: The `.dot` file generated automatically by the MathLang DRa part of the TEXMACS plugin. It consists of a description of the DRa Dependency Graph comprehensive view, see the righthand side of Figure 6.9.

6.4 The DRa Checker

The MathLang CGa+TSa annotated document can be passed to the MathLang checker to validate the well-formedness of the annotation. Similarly the DRa annotation can be automatically validated. In Section 4.7 we described the analysis of the DRa annotation which is performed in two phases, first on the Dependency Graph level and the second on the Graph of Logical Precedences level. In this section we present the informal algorithms that are used for checking the DRa. The informal algorithms presented in this section have been authored by this PhD student, where the implementation of similar algorithms have been carried on by C. Zengler. The work of C. Zengler involved the extension and improvement of the DRa towards the proof system Coq including the implementation of the checker for the DRa annotation.

The current development of the DRa checker allows to capture two kinds of failures: warnings and errors. At the moment we can check for:

1. a proof of an *unprovable* node (error),
2. missing proofs for a *provable* node (warning),
3. more than one proof for a *provable* node (warning),
4. an inconsistence annotation of mathematical/structural rhetorical roles (warning),
5. loops in the GoLP (error),

The checks for the first three points, 1–3, are performed on the DG level. Algorithm 1 given below, presents the checks for *unprovable* nodes. For every node of the dependency graph we check if the incoming relation is of type "justifies" and if the node type is "unprovable". If so, then the error is raised. This allows to capture situations where someone tries to prove an axiom or a definition (if the definition has been specified to be an unprovable node).

```
foreach node n of dependency graph G do
   foreach incoming edge e of n do
      (m, n) = getNodesOfEdge(e);
      /* The edge e is presented in form of RDF triples, e.g., e=(m, justifies, n). The function getNodes-
      OfEdge(e) returns nodes of the edge with preserved order. */
      if e is of type "justifies" && n is of type "unprovable" then
         raiseError('Node n.Id is unprovable but has annotated incoming relation "justifies":
         (m.Id, justifies, n.Id).');
      end
   end
end
```

Algorithm 1: checkUnprovableNodes(Graph G)

```
foreach node n of dependency graph G do
   foreach incoming edge e of n do
      (m, n) = getNodesOfEdge(e);
      /* The edge e is presented in form of RDF triples, e.g., e=(m, justifies, n). The function getNodes-
      OfEdge(e) returns nodes of the edge with preserved order. */
      role = getMathRoleOfNode(m);
      if e is of type "justifies" && n is of type "provable" && role is not "proof" then
         raiseWarning('Node n.Id has missing proof.');
      end
   end
end
```

Algorithm 2: checkMissingProofs(G)

The checks for inconsistent annotations of mathematical rhetorical roles (point number 4 from the above list), involves again the knowledge of "unprovable" and "provable" nodes. For instance, if a node n was annotated as *axiom* and later was annotated as a *conjecture* then the DRa checker would raise a warning, because *conjecture* could be a "provable" statement, whereas axiom is definitely "unprovable"

```
foreach node n of dependency graph G do
    A = Array(); /* Stores edges which "justify" node n */
    i = 0; /* The number of proofs of node n */
    foreach incoming edge e of n do
        if e is of type "justifies" && n is of type "provable" then
            n = n + 1;
            A.append((m,justifies,n));
            if n > 2 then
                raiseWarning('Node n.Id has more than one proof:');
                displayElementsOfArray(A);
            end
        end
    end
end
```

Algorithm 3: moreThanOneProof(G)

by default.

Similarly, let us assume that someone annotates a node n as *chapter* and later states it as *subsection* of the same parent node *n'* being *section*. Then the DRa should respond by a warning stating that the node n has different levels of depth in the document tree, i.e., *chapter* has a document depth level 1, whereas *section* has a document depth level 2. Although, this has not been implemented yet within the current DRa development and it involves the separation between assigning structural roles to nodes and providing their depth level in the document.

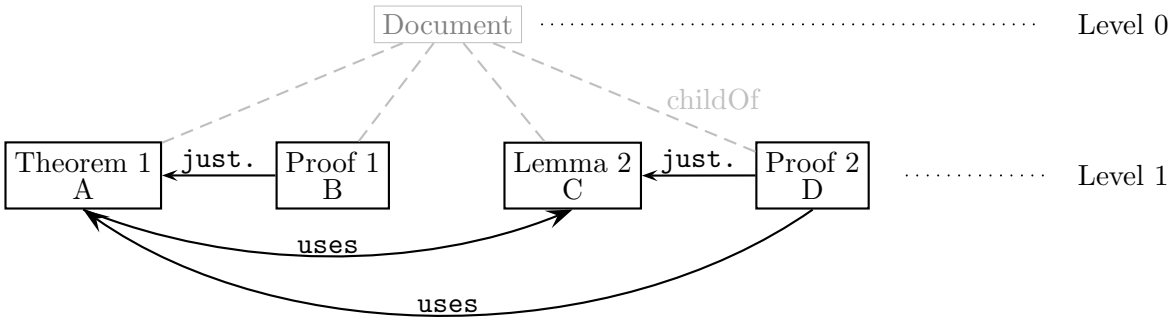


Figure 6.11: The example of a not recognised loop in the DRa dependency graph.

The example presents a possible annotation of a mathematical document with DRa elements. The righthand side of the example introduces the level information of the DRa annotated chunks of the document. The MathLang document can be presented as a tree (due to the XML concrete syntax and the TEXMACS internal tree structure). Hence each node of the tree is a child of (relation "childOf" from the child node to its parent) the node "document". However, in the DG graph we neither include the node "document" nor the relation "childOf", and we do not treat them as important. We present this node "document" and the relation "childOf" as a reference to the tree like structure of the document.

The check for loops in the GoLP of the DRa annotated document requires

the building of a *transitive closure* of the GoLP. This was explained previously in Section 4.7.2. Let us demonstrate this with a toy example. Consider the text annotated as the one presented in Figure 6.11.

This example demonstrates a cycle in the DRa annotated document. The problem is mainly, that Theorem 1 uses Lemma 2 but Proof 2 of Lemma 2 uses Theorem 1. This situation results in a deadlock. We have a cycle between nodes A, D, C with edges (A,D), (D,C), (C,A), see lefthand side of Figure 6.12. This scenario would probably not happen in any mathematical paper or book, but it could appear in the MathLang annotated document especially if the annotation is done by pupils or students, and the annotated document consists of hundreds of pages.

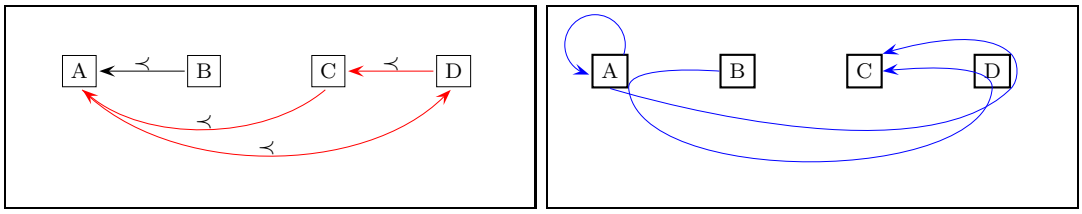


Figure 6.12: The GoLP graph of the example of figure 6.11 and the Transitive Closure edges of the GoLP graph (left GoLP, right Transitive Closure edges).

So the question is, how do we find a cycle in a big document annotated with the DRa entities?

This was already discussed in Section 4.7.2, but we recall the algorithm here. First we automatically extract the dependency graph from the DRa annotated document, see Figure 6.11 for our example. Then we automatically transform the DG into graph of logical precedences by applying the transformation function 4.13. At this stage we might notice the cycle in the GoLP, see for instance the lefthand side of Figure 6.12 where the cycle has been drawn using a different color. Now we build the *transitive closure* of the GoLP using for instance the Roy-Warshall's algorithm [Roy59, War62]. On the righthand side of Figure 6.12 we presented the extra edges created while building the *transitive closure* of the GoLP for the example from Figure 6.11. At this stage we check if the *transitive closure* does not have any loops, or in other words there is no reflexive edge for any node. In the case of our example the loop is visible immediately for node A. Once the loop is discovered the DRa checker raises an error and displays all paths that lead to the deadlock situation.

6.5 Conclusion

In this chapter we summarised our development and implementation of tools for the MathLang DRa aspect. Section 6.1 discussed the XML concrete syntax of the DRa. We compared the *plain syntax* with the *concrete XML syntax* of the DRa, in the form of a table, see Table 6.1. We also provided the grammar of the DRa as an RELAX NG scheme, see Listing 6.1. We used for that the compact syntax of the RELAX NG which can be easily transformed into its XML equivalent syntax.

The next section presented our implementation on the TEXMACS side. We discussed the development of the DRa part of the MathLang plugin for TEXMACS. We presented the DRa macros for TEXMACS and the implementation using the SCHEME language. This development allows the annotation of the DRa aspect of a mathematical document using TEXMACS and MathLang plugin.

The following section presented the transformation tools for a MathLang DRa annotated document. We presented the XSLT stylesheets that we implemented to allow the automatic transformation of the DRa annotation into graph presentation, the Dependency Graph and the Graph of Logical Precedences. We also implemented the transformation functions within the MathLang plugin, which provides the automatic generation and display of graphs directly from the MathLang-DRa menu in TEXMACS. We discussed our use of the DOT language to describe graphs and to build Postscript files displaying these graphs.

Finally in the last section, we presented a brief overview of the informal algorithm for checking the well-formedness of DRa annotations.

Chapter 7

Future Developments, Related Works and Conclusion

In this chapter we discuss current and future development of MathLang (Section 7.1). In addition, we reflect on related works and short comparison of the DRa aspect to other systems in Section 7.2. Finally, in the last Section 7.3 we give a conclusion to the thesis.

7.1 MathLang's Current and Future Development

7.1.1 Current Development

The MathLang project, started in 2000 by F. Kamareddine and J.B. Wells, provides a novice approach to the computerisation of mathematical documents. At the heart of this approach is the gradual computerisation via the annotation of different "aspects" of a mathematical document. The final goal of the MathLang annotation is the partial/full formalisation of the same document for different *proof systems*. The current development of the framework provides a computer-assisted authoring tool for annotating these "aspects". Moreover, while annotating a document with MathLang aspects, this tool does not require from the user any expertise skills in computer-based formalisation in a specific *proof system*.

The ground bases of the MathLang framework were studied by M. Maarek who developed, under the supervision of F. Kamareddine and J.B. Wells, the CGa and TSa "aspects". These aspects are extensively reported in his PhD thesis [Maa07]. The development of the TSa aspect involved also the collaboration of R. Lamar, another research student in the MathLang project.

The main research and development of K. Retel was oriented towards the Document Rhetorical aspect (DRa) of mathematical documents. This thesis reports on that aspect and also reflects on the gradual computerisation of a CML document towards full formalisation in the Mizar language.
The forthcoming PhD thesis of R. Lamar relates to the development of the MathLang path towards the Isabelle *proof system*. Another research student of the MathLang project, C. Zengler, reviewed the DRa aspect and MathLang annotated document towards the Coq *proof system*. He also implemented the DRa checker and provided another view – a GoTO (*a graph of textual order*) – of the DRa annotation and graphs transformation, see [KWZ08]. C. Zengler contributed to the MathLang framework with rewriting the MathLang CGa and TSa checker in the modern programming language JAVA. Previous implementation (in OCaml) of the MathLang checker was due to M. Maarek.

7.1.2 Future Development

The future development discussion has been well thought in the MathLang group. Most of the ideas has been discussed in many occasions during weekly group meetings. They have been initially and partially described in MathLang papers, but were then put together and summarised in the thesis of M. Maarek. In this section we recall them and shortly present them to the reader nin order to provide a general idea of further MathLang development challenges.

7.1.2.1 Extending DRa

The DRa encoding system is a part of the ongoing MathLang project. As future work, we need to concentrate on the evaluation and improvement of the DRa system and to test it on a number of bigger examples. We also need to work further on the DRa ontology and to refine the instances of the class StructuralRhetoricalRole. Namely, we need to separate the depth level of structural units labels from the actual meaning of a unit. For instance "section" and "subsection", for the representation purposes, differ only in the embedded relation. Therefore we have to investigate how the depth level can be incorporated within the DRa ontology.

One of the advantages of the DRa encoding is modularization. We still need to investigate how this modularization should work, and how we can actually reuse the annotated distinct chunks of text in other documents. In addition to that, we need to research how we can relate some parts of a document in another document.

This would require a review of both aspects CGa and DRa.

We also need to investigate how a mathematician could add his own intended relation to the DRa system. For instance, he might want to add the explanationOf relation which could be used to express the statement: "this example is an explanation of such definition", which might be written in the RDF triples format like that: $(example, \mathsf{explanationOf}, definition)$. We have to incorporate this kind of possibilities within the DRa markup system.

7.1.2.2 Automatisation of CGa+TSa encoding

At the current development stage, the MathLang user is required to annotate and specify grammatical role (in a CGa aspect) for each symbol in every CML statement. In addition to that, the user has to write a signature (symbol name, number of input and output types together with their grammatical roles) for that symbol in the preamble of the document. In most of the cases, a lot of symbols are used more than once in a document, which still have to be annotated explicitly by the user. This makes the annotation process time consuming. We believe that this process can be done with a computer assistance and much faster if we develop specific software. Let us assume that the user annotates only the first instance/usage of a symbol and provides a signature of that symbol in the preamble. If this symbol has been used further in a document, the proposed tool should be able to parse the symbol and annotate it in the whole document accordingly to the signature provided in the preamble. Such a tool needs to be able to parse formulas of the original CML document, which is possible due to the presentation of the document in $\mathrm{TEX_{MACS}}$ and XML format. Of course this annotation will be designed for a symbol whose spelling does not change across the whole document.

7.1.2.3 Informal Justification aspect (IJa)

The CGa aspect does not provide a direct way of linking reasoning statements or references to existing theorems. In the CML document it is usually done via the use of a number of words like: *hence*, *since*, *then* or *by*. This aspect will provide a way to annotate those references and link between statements.

In addition, the CGa local scoping does not provide a way to differentiate "considerations", "assumptions" or "hypothesis" of declarations. The meaning of these textual components will be exposed by this aspect. The relations between them might be expressed with theb extended DRa. Such provided annotation will

be automatically checked and will complete the initial CGa structure into a more formal document.

7.1.2.4 Meta-logic aspect (MLa)

As described by M. Maarek in his thesis, another possible aspect for MathLang is concerned with "meta-logic". A text that has been annotated with CGa+DRa+IJa contains the logic and semantic information that a CML document could have. Such text could possibly still contain holes in the reasoning and logic, but this purely depends on the author writing style. This aspect will provide a generic language to *describe* some of the existing logical frameworks. This would allow the user to chose the final destination framework he wants to work with (e.g., the Tarski-Grothendieck Set Theory – the Mizar system, the Zermelo-Frankel Set Theory – the Isabelle system, the Calculus of Inductive Constructions – the Coq system, etc.). A text annotated with the Meta-logic aspect will be analysed under the well-formation and well-use of the aspect. The proof itself will be checked in the final logical framework.

7.1.2.5 Automatisation while building the Mizar document

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. It is important to research further automatisation of building a Mizar *Text-Proper* skeleton in the TEXMACS editor. Furthermore, it is useful to have an additional tool for supporting the transformation of the CGa annotated statements into the Mizar formula level. It is also important to study in depth the stage where a Mizar FPS version is fully formalised into Mizar. A number of issues need to be investigated to support building Mizar statements:

- 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.

7.2 Related works

Many studies have been carried on the structure of documents. For example, the Text Encoding Initiative Guidelines (`http://www.tei-c.org/`) are international standards that enable the representation of a variety of literary and linguistic texts. DocBook (`http://www.docbook.org`), provides a system for writing a structured document using XML. Another tool is OMDoc (developed by M. Kohlhase [Koh06a] - see below). These systems allow to separate/divide a document into a number of structural components (sections or mathematical assertions) which can be annotated in the computerised version. Our proposed markup system is simpler and is concentrated only on the annotation of the narrative structure of mathematical documents, whereas others are more oriented towards capturing other subtleties of the document. We believe that separating the concerns during computerisations can play a very helpful role in developing computer tools that can aid various levels of computerisation/formalisation.

7.2.1 OMDoc vs DRa – a short comparison

OMDoc presents mathematical knowledge on three levels: the object and formula level, the statement level, and the theory level. What is made explicit by the DRa markup, is similar to the statement level and partly to the theory level in the OMDoc system. The OMDoc markup distinguishes the knowledge elements of a theory into constitutive ones like symbols, axioms, and definitions (which present the essence of the annotated theory) and non-constitutive ones such as assertions, their proofs, examples (which illustrate properties and attributes of mathematical objects determined by the constitutive statements). This shows a different approach to annotating the same knowledge. The aim of introducing the DRa is to be able to catch and store the narrative structure of the text, and simultane-

ously allow to stay as close as possible to the original document and the style it was written in. Therefore, on the DRa markup we do not distinguish constitutive or non-constitutive statements. We recognize only one class of elements, called StructuredUnit, and we distinguish the roles they play in mathematical knowledge representation.

Therefore, the purposes/aims of OMDoc and DRa are different. All instances of the class MathematicalRhetoricalRole in the DRa ontology, are presented as disjoint classes in the OMDoc ontology [Lan06]. "axiom" is an ontology class in the OMDoc ontology, whereas in the DRa it is expressed as an instance (individual) of the class MathematicalRhetoricalRole. This particular name "axiom" expresses a role of the text labeled by that name, and hence in the DRa ontology we annotate it by stating the property hasMathematicalRhetoricalRole whose range value is an appropriate instance (i.e., "axiom") of a class MathematicalRhetoricalRole.

```
<definition xml:id="node-D1.def">
  <CMP>A subset A ⊂ R is inductive if [...]
<assertion xml:id="thm-T1" type="theorem">
  <CMP>Let J be a subset of Z+ [...]
<proof xml:id="proof-PT1" for="#thm-T1">
  <CMP>J is inductive so J contains [...]
```

Both annotation systems OMDoc and DRa allow to markup dependencies between statements. In the OMDoc file format they are implemented by means of the `for` attribute to OMDoc's elements (e.g., `<proof for="#id-of-assertion">`). A possible encoding of a part of our main example shown in Figure A.1 in OMDoc is sketched[1] above. Within the DRa system we annotate the relation as an RDF triple, and it might be expressed in the MathLang internal file using any kind of XML-RDF recommendations.

The other and main advantage of DRa over OMDoc is a possible analysis of the dependency graph and the GoLP, which are automatically built from the performed annotation. This analysis allows to check the annotation of the narrative/rhetorical aspect of a document (see Section 4.7). Although OMDoc gives a lot of elements and constructions that can be used to structure mathematical documents, these allow the user some software compatibility but no validation yet. The DRa annotation system gives the user a validation tool making it possible to analyse the well-formedness/encoding of the rhetorical aspect of a document.

[1]For readability and brevity, we show only the opening tag of each XML element for most elements; we use indentation to express nesting.

7.2.2 Other works related to DRa

Nakagawa, Nomura and Suzuki [NS06, NNS04] present a method to express the logical structure of a document and hyperlinks between chunks of mathematical text that enhance the readability of a document and the navigation throughout the text. That method detects the logical structure of a text and several types of hyperlinks from printed mathematical documents. Our approach differs in the sense that we propose an annotation system that allows to express such logical structure and hyperlinks/relations while authoring a document. Moreover, we use the DG achieved from the annotation to build a formal document skeleton (as we have done in Mizar and can be done in other systems).

7.2.3 Mizar and WTT comparison work

Geleijnse [Gel04] 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 in an easier manner 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 allow 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 2.5 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 2.2) 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 [JN04]. 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's 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.

7.3 Conclusion

This thesis presents our MathLang approach to encoding mathematics on computers. This approach defends the idea that computerisation should come before any formalisation. More importantly, we believe that the computerisation process can be divided into a number of steps, where some steps might be performed without requiring an expert computer programming skills. We show in the thesis 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 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.

This thesis presents our approach to computerise the narrative aspect of mathematical texts. We demonstrate in the thesis the MathLang philosophy. We describe MathLang *aspects*, formal languages and annotation systems which are used to annotate CML documents with MathLang. We built a DRa ontology which describes formally the domain of narrative/structural representations of mathematical knowledge in a document. The ontology allows to share a common understanding of the structure of the represented knowledge among other people and software agents. The ontology separates a domain knowledge (DRa) from the operational knowledge – the actual annotation. By using the ontology we annotate/mark up our main example shown in Figure A.1.

We give the meaning behind the DRa annotation and give automated tools which generate different representations of the document structure. We show how the encoded Document Rhetorical aspect annotation could be validated for checking the well-formedness of the annotation. We also express which mistakes made during

annotation we are able to automatically catch. Finally we demonstrate how the DG and the Graph of Logical Precedences are used to build the skeleton of Mizar *Text-Proper*.

We demonstrate in this thesis our novice approach to the formalisation path. This path starts with a CML document, then follows annotation via MathLang aspects. Then the Mizar expert builds a skeleton of the Mizar FPS and parts of the "environment" of the Mizar FPS, reusing the MathLang annotation of the CML document and following hints and rules that we developed and described in this thesis. Finally such document is moved towards a proper Mizar FPS and into a full formalisation in Mizar. The thesis also describes a short comparison between MathLang constructs and Mizar counterparts. It also demonstrates hints and rules that can be used for building a semi-automatic tool for transforming a MathLang annotated CML document into Mizar FPS.

Appendix A

Original, DRa-annotated text and Mizar formalisation for number of examples

A.1 Moller example CML+DRa

The original text of the given example is taken from J.M. Möller's notes [Mol07] regarding general topology and is reproduced on the left hand side of Figure A.1. The right hand side of the figure shows the automatically generated dependency graph for the text, previously annotated with the MathLang Document Rhetorical aspect. The relations between parts of the text are represented by visible arrows and graph nodes have specified (but not visible) mathematical or structural rhetorical roles.

2. The integers and real numbers

We shall **assume** that real numbers $\mathbf{R}$ exists with all the usual properties: $(\mathbf{R}, +, \cdot)$ is a field, $(\mathbf{R}, +, \cdot, <)$ is an ordered field, $(\mathbf{R}, <)$ is a linear continuum (1.15).
What about $\mathbf{Z}_+$?

DEFINITION 1.17..

A subset $A \subset \mathbf{R}$ is inductive if $1 \in A$ and $a \in A \Longrightarrow a + 1 \in A$

There are inductive subsets of $\mathbf{R}$, for instance $\mathbf{R}$ itself and $[1, \infty)$.

DEFINITION 1.18..

$\mathbf{Z}_+$ is the intersection of all inductive subsets of $\mathbf{R}$.

We have that $1 \in \mathbf{Z}_+$ and $\mathbf{Z}_+ \subset [1, \infty)$ because $[1, \infty)$ is inductive so $1 = \min \mathbf{Z}_+$ is the smallest element of $\mathbf{Z}_+$.

THEOREM 1.19.. *(Induction Principle)*

Let J be a subset of $\mathbf{Z}_+$ such that

$$1 \in J \text{ and } \forall n \in \mathbf{Z}_+ : n \in J \Longrightarrow n + 1 \in J$$

Then $J = \mathbf{Z}_+$.

Proof 1.

J is inductive so J contains the smallest inductive set $\mathbf{Z}_+$.

THEOREM 1.20..

Any nonempty subset of $\mathbf{Z}_+$ has a smallest element.

Before the proof, we need a lemma.
For each $n \in \mathbf{Z}_+$, write

$$S_n = \{x \in \mathbf{Z}_+ \mid x < n\}$$

for the set of positive integers smaller than n (the *section* below n). Note that $S_1 = \emptyset$ and $S_{n+1} = S_n \cup \{n\}$.

LEMMA 1.21..

For any $n \in \mathbf{Z}_+$, any nonempty subset of S_n has a smallest element.

Proof 2.

Let $J \subset \mathbf{Z}_+$ be the set of integers for which the lemma is true. It is enough (1.19.) to show that J is inductive. $1 \in J$ for trivial reason that there are no nonempty subsets of $S_1 = \emptyset$. Suppose that $n \in J$. Consider a nonempty subset A of S_{n+1}. If A consists of n alone, then n=min$A$ is the smallest element of A. If not, A contains integers $< n$, and then min$(A \cap S_n)$ is the smallest element of A. Thus $n + 1 \in J$.

Proof of Theorem 1.20 1.

Let $A \subset \mathbf{Z}_+$ be any nonempty subset. The intersection $A \cap S_n$ is nonempty for some n, so it has a smallest element (1.21.). This is also the smallest element of A.

[...]

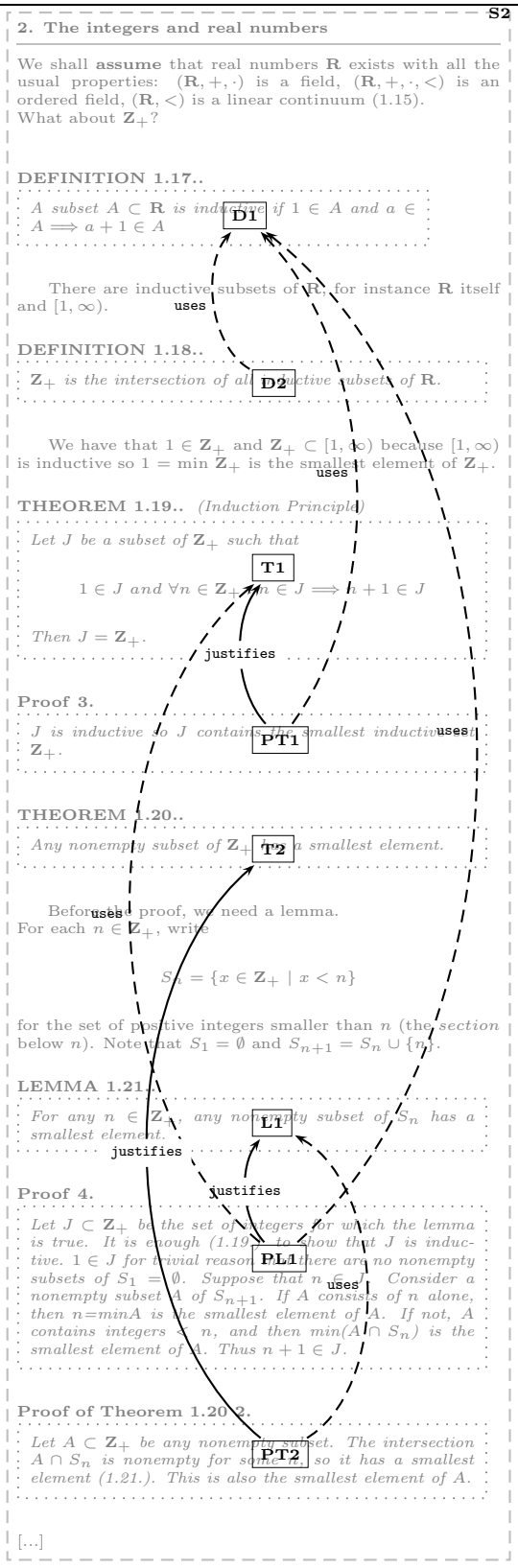


Figure A.1: Fragment of J.M. Möller's text ([Mol07, Chapter III, §2]) with and withoput dependency graph.

A.2 Pythagoras' theorem example by G.H. Hardy and E.M. Wright

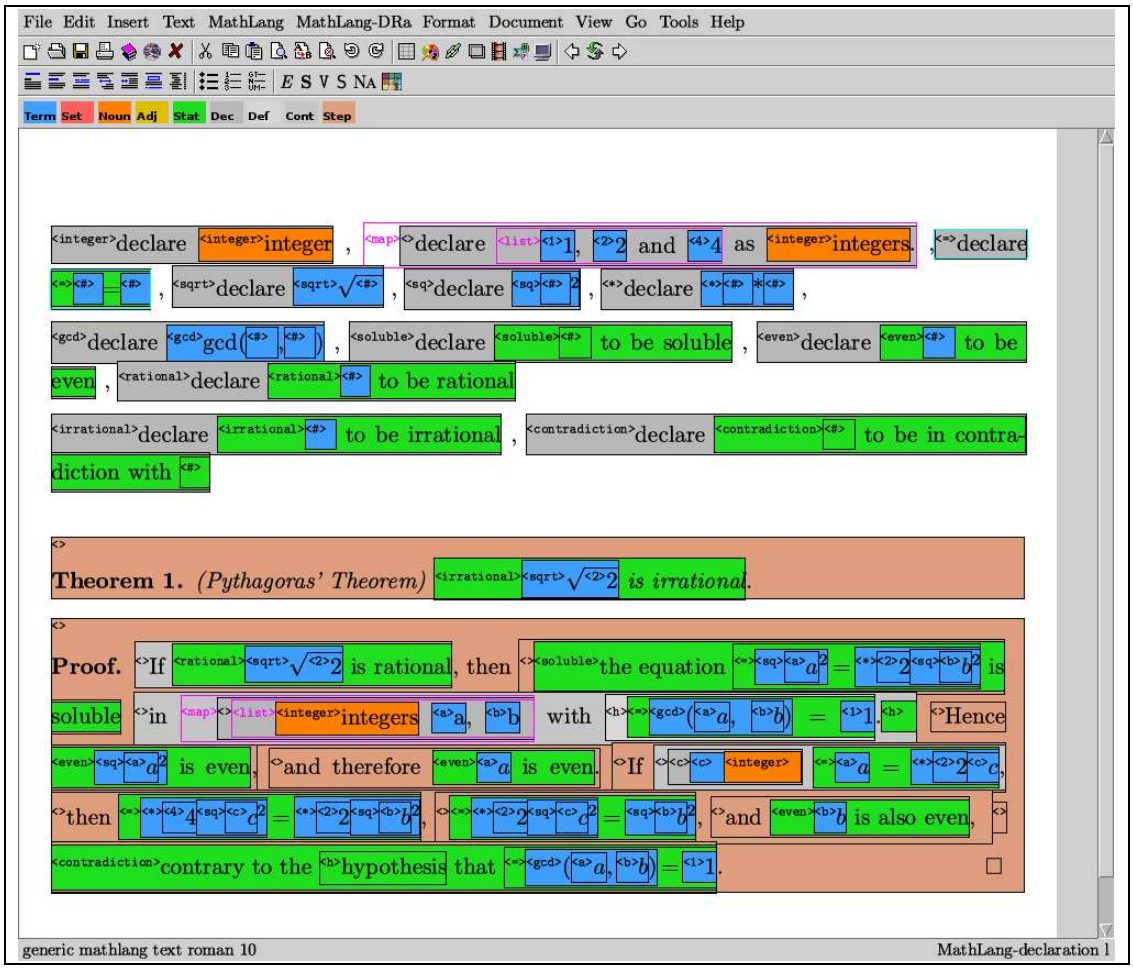


Figure A.2: The MathLang CGa+TSa version with MathLang interpretation of the original document from Figure 5.3 – the MathLang interpretation is attached to the grammatical role of each element.

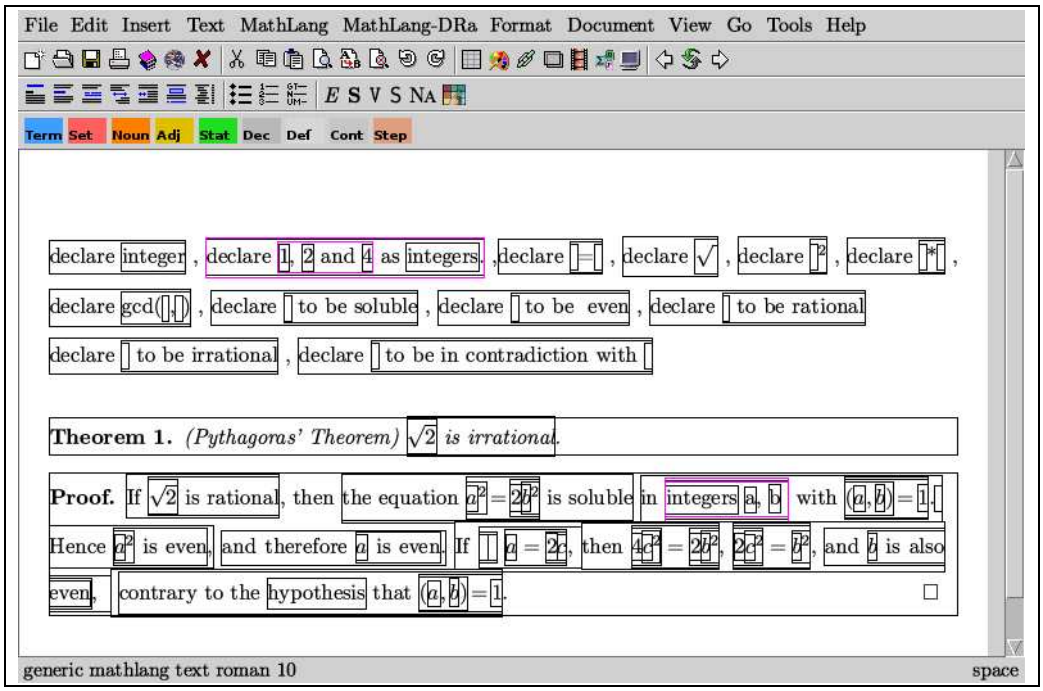


Figure A.3: The MathLang CGa+TSa version with boxes and without colours of the original document from Figure 5.3.

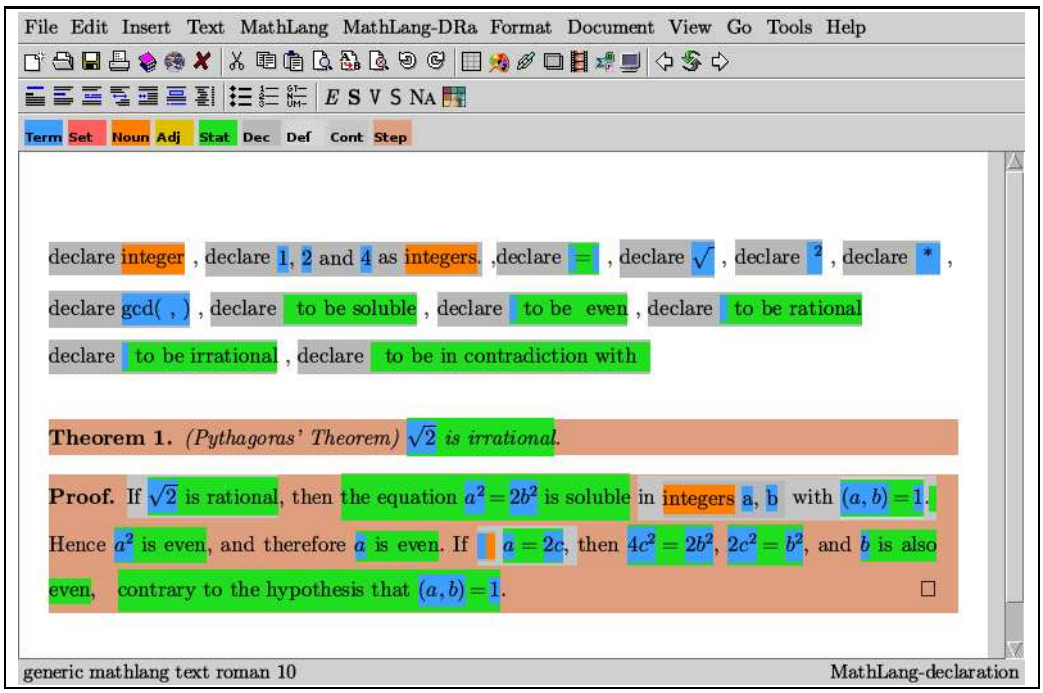


Figure A.4: The MathLang CGa+TSa version with colours and without boxes of the original document from Figure 5.3.

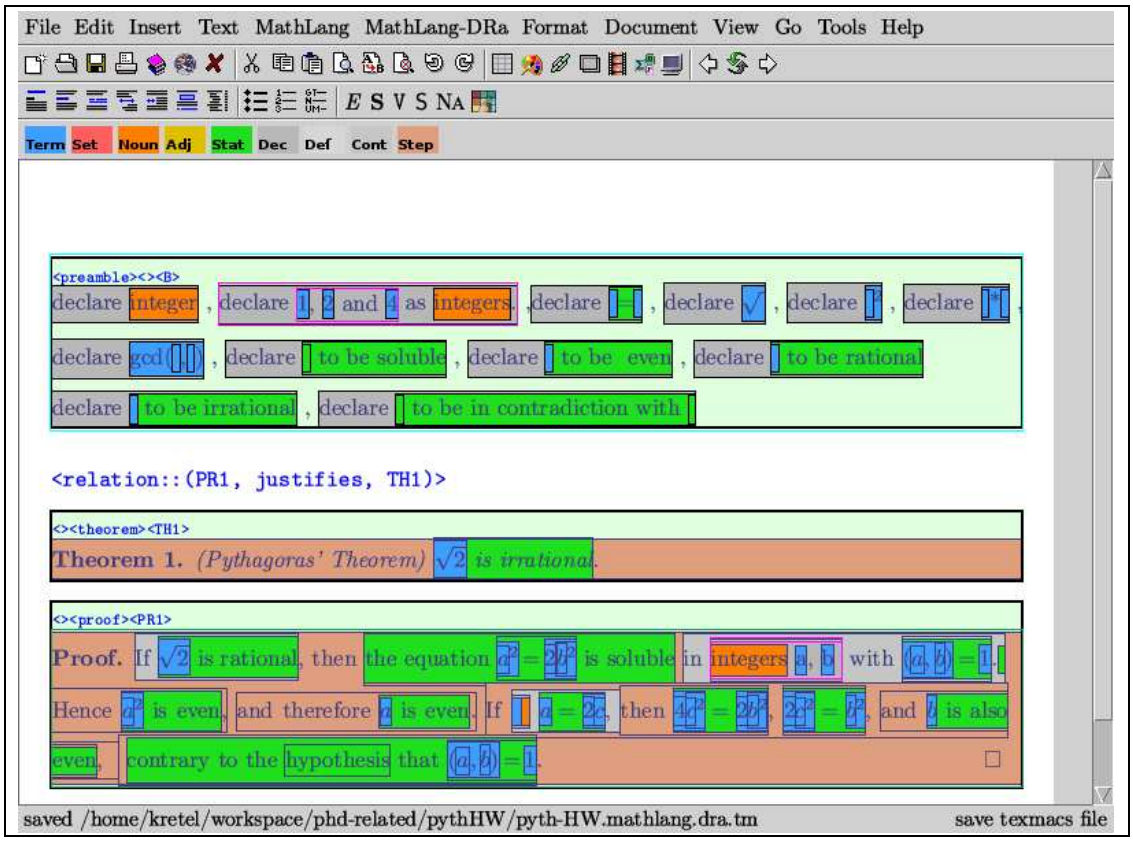


Figure A.5: The MathLang CGa+TSa+DRa version of the original document from Figure 5.3.

```
(* Encoding of Pythagoras Theorem Hardy and Wright version. *)
(* ULTRA Group *)

{
   1(): term;
   2(): term;
   4(): term;
   contradiction(stat): stat;
   = (term,term):stat;
   sq (term):term;
   sqrt (term): term;
   * (term,term):term;
   gcd(term,term):term;

   soluble(stat):stat;
   even(term):stat;
   rational(term): stat;
   irrational(term): stat;
   integer(): noun;
};


Th := irrational(sqrt(2));

{
 { rational(sqrt(2)); } |>
 {
   {a:integer; b:integer;  =(gcd(a,b),1);} |>
   {
     soluble(=(sq(a),*(2,sq(b))));
     even(sq(a));
     even(a);
     {c:integer; =(a,*(2,c));} |>
     {
       =( *(4,sq(c)) , *(2,sq(b)));
       =( *(2,sq(c)), sq(b));
       even(b);
       label #L1 =(gcd(a,b) ,2);
       {ref #L1 |> contradiction(=(gcd(a,b),1)); };
     };
   };
 };
};
```

Listing A.1: The MathLang CGa *plain syntax* version of the proof of Pythagoras' theorem by G.H. Hardy and E.M. Wright.

```
environ
 vocabularies SQUARE_1, IRRAT_1, RAT_1, ARYTM,
   INT_1, ARYTM_3, MATRIX_2, ORDINAL2;
 notations SQUARE_1, RAT_1, IRRAT_1, ORDINAL1, XREAL_0,
   INT_1, INT_2, XCMPLX_0, ABIAN;
 constructors NUMBERS, XXREAL_0, XREAL_0, SQUARE_1, RAT_1,
   INT_2, XCMPLX_0, POWER, ABIAN;
 registrations XREAL_0, REAL_1,
   INT_1, NAT_1, SQUARE_1;
 requirements BOOLE, SUBSET, NUMERALS, REAL, ARITHM;

begin :: Proof of the Irrationality of square root of 2,
      :: original approach by Hardy and Wright.

theorem TH1:  :: Theorem statement required.
  sqrt 2 is irrational
proof
  :: Proof of the theorem statement.
  assume sqrt 2 is rational;
  consider a,b being Integer such that
  a^2 = 2 * b^2 and
  a,b are_relative_prime;
::>                    *4
  a^2 is even;
::>         *4
  a is even;
::>       *4
  consider c being natural number such that
  a = 2*c;
::>     *4
  4*c^2 = 2*b^2;
::>           *4
  2*c^2 = b^2;
::>         *4
  b is even;
::>       *4
  thus contradiction;
::>                *1
end;
::> 1: It is not true
::> 4: This inference is not accepted
```

Listing A.2: The Mizar FPS version of the proof of Pythagoras' theorem by G.H. Hardy and E.M. Wright.

```
environ
 vocabularies SQUARE_1, IRRAT_1, RAT_1, ARYTM, INT_1, ARYTM_3, MATRIX_2,
   ORDINAL2, INT_2, SUBSET_1, RELAT_1;
 notations NUMBERS, SQUARE_1, RAT_1, IRRAT_1, ORDINAL1, XREAL_0, INT_1, INT_2,
   XCMPLX_0, ABIAN, NAT_1, PEPIN, XXREAL_0, SUBSET_1;
 constructors NUMBERS, XXREAL_0, XREAL_0, SQUARE_1, RAT_1, INT_2,
   XCMPLX_0, POWER, ABIAN, PEPIN, REAL_1, ARYTM_2;
 registrations XREAL_0, REAL_1, INT_1, NAT_1, SQUARE_1, NUMBERS, ORDINAL1,
   XXREAL_0;
 requirements BOOLE, SUBSET, NUMERALS, REAL, ARITHM;
 theorems RAT_1, SQUARE_1, NAT_1, XCMPLX_1, PYTHTRIP, ABIAN, INT_2,
   INT_1, REAL_2, XREAL_1, XXREAL_0, ORDINAL1;
 definitions XCMPLX_0, SQUARE_1;

begin :: Proof of the Irrationality of square root of 2,
      :: original approach by Hardy and Wright.

LocalTH1:
 for k,l being Integer st not k,l are_relative_prime holds
    ex x being Integer st x = k gcd l & x <> 1
  proof
    let k,l be Integer;
    assume not k,l are_relative_prime;
    then A1: k gcd l <> 1 by INT_2:def 4;
    thus thesis by A1;
  end;

LocalTH2:
  for p being Rational  holds
  ex a,b being Integer st b <> 0 & p = a/b & a,b are_relative_prime
proof
  let p be Rational;
  consider m being Integer, k being Element of NAT such that
  A0: k <> 0 and
  A1: p = m/k and
  A2: for n being Integer, l being Element of NAT
      st l <> 0 & p = n/l holds k <= l by RAT_1:25;
  m,k are_relative_prime
  proof
    assume not thesis;
    then consider x being Integer such that
    B1: x = m gcd k & x <> 1 by LocalTH1;
    B2: x divides m & x divides k by B1,INT_2:32;
    then B0: x <> 0 by INT_2:10,A0;
    consider l1 being Integer such that
    B3: m = x * l1 by INT_1:def 9,B2;
    consider l2 being Integer such that
    B4: k = x * l2 by INT_1:def 9,B2;
    B5: p = (x/x) * (l1/l2) by XCMPLX_1:77,B3,B4,A1
         .= 1 * (l1/l2) by B0,XCMPLX_1:60
         .= l1/l2 ;
    reconsider x as Element of NAT by B1,INT_2:29;
    B9: l2 <> 0 by B4,A0;
    B8: l2 > 0
```

```
    proof
      assume not thesis;
      then l2 <= 0 & x > 0 by NAT_1:3, B0;
      then k <= 0 by B4, XREAL_1:133;
      hence contradiction by A0,NAT_1:3;
      end;
    then l2 in NAT by INT_1:16;
    then reconsider l2 as Element of NAT;
    k > l2
    proof
      assume D0: not thesis;
      then k <= l2;
      per cases by XXREAL_0:1,D0;
      suppose k < l2;
        then k/l2 < l2/l2 by XREAL_1:76,B8;
        then k /l2 < 1 by XCMPLX_1:60,B9;
        then x/(l2/l2) < 1 by B4,XCMPLX_1:78;
        then x /1 < 1 by XCMPLX_1:60,B9;
        then x <= 0 or x = 0+1 by NAT_1:26;
        hence contradiction by B0,B1,NAT_1:3;
      end;
      suppose k = l2;
        hence contradiction by B1,B4,B9,XCMPLX_1:7;
      end;
    end;
    hence contradiction by A2,B5,B8;
  end;
  hence thesis by A1,A0;
end;


theorem TH1:  :: Theorem statement required.
  sqrt 2 is irrational
proof
  :: Proof of the theorem statement.
  assume sqrt 2 is rational;
  then consider a,b being Integer such that
  A1: b <> 0 and
  A2: sqrt 2 = a/b and
  A3: a,b are_relative_prime by LocalTH2;
  A4: b^2 <> 0 by A1,SQUARE_1:74;
  0 <= 2 by NAT_1:18; then
  2 = (a/b)^2 by A2,SQUARE_1:def 4
   .= (a*a)*(b"*b")
   .= (a*a)*(b*b)" by XCMPLX_1:205
   .= a^2/b^2;
  then A6: a^2 = 2*b^2 by A4,XCMPLX_1:88;
  then a^2 is even by ABIAN:def 1;
  then a is even by PYTHTRIP:2;
  then consider c being Integer such that
  A8: a = 2*c by ABIAN:def 1;
  4*c^2 = (2*2)*c^2
        .= 2^2*c^2 by SQUARE_1:def 3
        .= 2*b^2 by A8,SQUARE_1:68,A6;
  then A9: 4*c^2 = 2*b^2;
```

```
  2*(2*c^2) = (2*2)*c^2
            .= 2*b^2 by A9;
  then 2*c^2 = b^2 by XCMPLX_1:5;
  then b^2 is even by ABIAN:def 1;
  then b is even by PYTHTRIP:2;
  then ex j being Integer st b = 2*j by ABIAN:def 1;
  then 2 divides a & 2 divides b by A8,INT_1:def 9;
  then A11: 2 divides a gcd b by INT_2:33;
  a gcd b = 1 by A3,INT_2:def 4;
  hence contradiction by A11,INT_2:17;
end;
```

Listing A.3: The full formalisation in Mizar of the proof of Pythagoras' theorem by G.H. Hardy and E.M. Wright.

```
environ
 vocabularies SQUARE_1, IRRAT_1, RAT_1, ARYTM, INT_1, ARYTM_3, MATRIX_2,
   ORDINAL2, INT_2, SUBSET_1, RELAT_1;
 notations NUMBERS, SQUARE_1, RAT_1, IRRAT_1, ORDINAL1, XREAL_0, INT_1, INT_2,
   XCMPLX_0, ABIAN, NAT_1, PEPIN, XXREAL_0, SUBSET_1;
 constructors NUMBERS, XXREAL_0, XREAL_0, SQUARE_1, RAT_1, INT_2,
   XCMPLX_0, POWER, ABIAN, PEPIN, REAL_1, ARYTM_2;
 registrations XREAL_0, REAL_1, INT_1, NAT_1, SQUARE_1, NUMBERS, ORDINAL1,
   XXREAL_0;
 requirements BOOLE, SUBSET, NUMERALS, REAL, ARITHM;
 theorems RAT_1, SQUARE_1, NAT_1, XCMPLX_1, PYTHTRIP, ABIAN, INT_2,
   INT_1, REAL_2, XREAL_1, XXREAL_0, WSIERP_1, ORDINAL1;
 definitions XCMPLX_0, SQUARE_1;

begin :: Proof of the Irrationality of square root of 2,
      :: original approach by Hardy and Wright.

LocalTH2:
  for p being Rational  holds
  ex a,b being Integer st b <> 0 & p = a/b & a,b are_relative_prime
  proof
    let p be Rational;
    take a=numerator(p),b=denominator(p);
    thus thesis by RAT_1:29,37,WSIERP_1:29;
  end;

theorem :: Phytagoras' theorem
  sqrt 2 is irrational
proof
  assume sqrt 2 is rational;
  then consider a,b being Integer such that
  A1: b <> 0 and
  A2: sqrt 2 = a/b and
  A3: a,b are_relative_prime by LocalTH2;
  A4: b^2 <> 0 by A1,SQUARE_1:74;
  0 <= 2 by NAT_1:18; then
  2 = (a/b)^2 by A2,SQUARE_1:def 4
   .= (a*a)*(b"*b")
   .= (a*a)*(b*b)" by XCMPLX_1:205
   .= a^2/b^2;
  then A6: a^2 = 2*b^2 by A4,XCMPLX_1:88;
  then a^2 is even by ABIAN:def 1;
  then a is even by PYTHTRIP:2;
  then consider c being Integer such that
  A8: a = 2*c by ABIAN:def 1;
  A9: 4*c^2 = (2*2)*c^2
           .= 2^2*c^2 by SQUARE_1:def 3
           .= 2*b^2 by A8,SQUARE_1:68,A6;
  2*(2*c^2) = (2*2)*c^2
           .= 2*b^2 by A9;
  then 2*c^2 = b^2 by XCMPLX_1:5;
  then b^2 is even by ABIAN:def 1;
  then b is even by PYTHTRIP:2;
  then ex j being Integer st b = 2*j by ABIAN:def 1;
```

```
  then 2 divides a & 2 divides b by A8,INT_1:def 9;
  then A11: 2 divides a gcd b by INT_2:33;
  a gcd b = 1 by A3,INT_2:def 4;
  hence contradiction by A11,INT_2:17;
end;
```

Listing A.4: Another approach to the full formalisation in Mizar of the proof of Pythagoras' theorem by G.H. Hardy and E.M. Wright. This formalisation approach reduces the proof of the local theorem labelled "LocalTH2" to 3 lines. It was proposed by A. Naumowicz during a discussion of the author of this thesis on the Mizar "developer-forum".

A.3 Pythagoras' theorem example by H. Barendregt

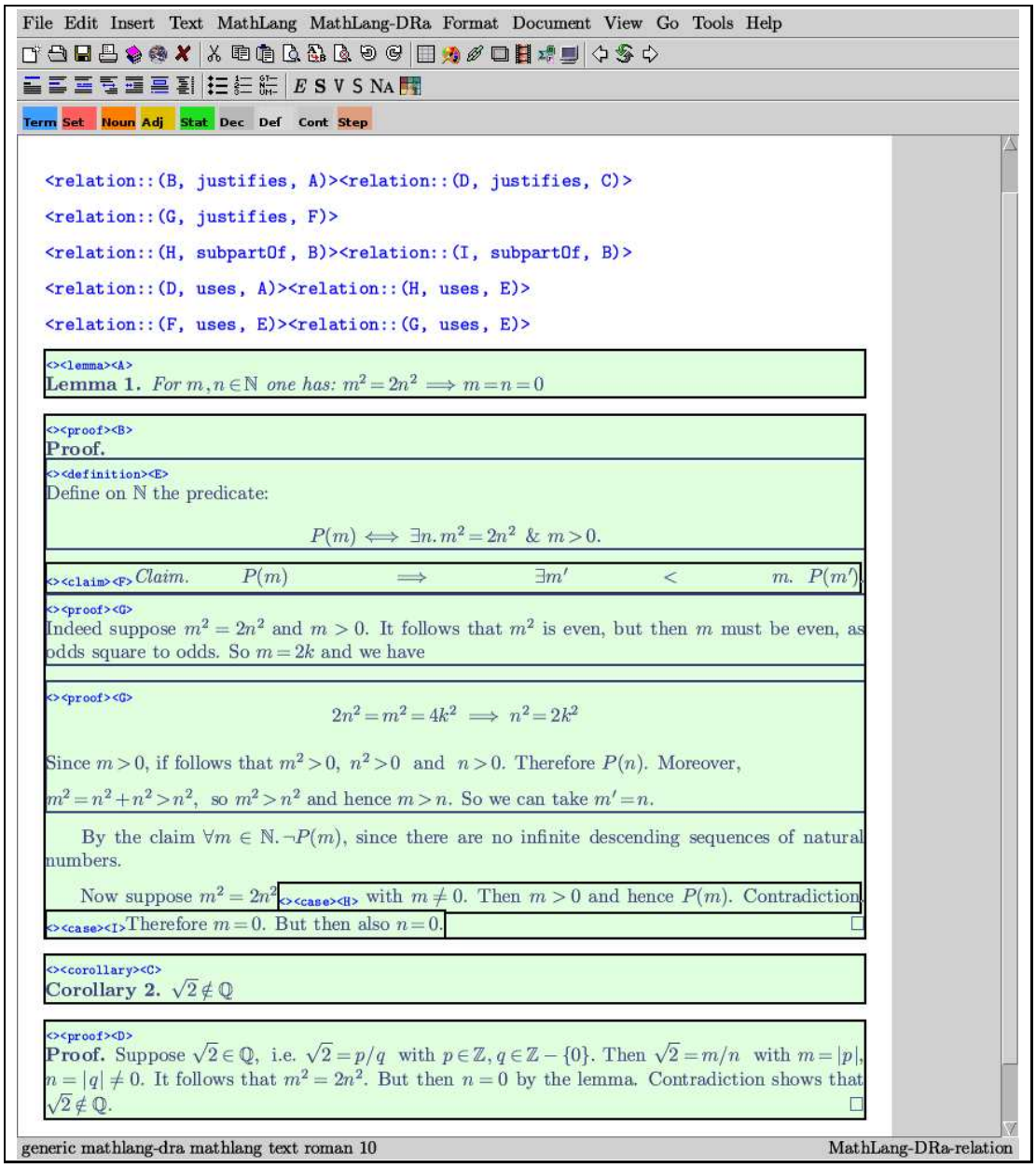


Figure A.6: The MathLang DRa annotation (version with DRa annotation displayed) of the original document from Figure 4.6.

```
(* Henk Barendregt version of Pythagoras Theorem proof.*)
(* ULTRA Group *)

{ not(stat): stat;
  and(stat,stat): stat;
  implies(stat,stat): stat;
  contradiction(): stat;
  forall(dec(‘x), stat) : stat;
  exists(dec(‘x), stat) : stat;
  0: term; 2: term; 4: term;
  N: set; Q: set; Z: set;
  =(term,term): stat;
  neq(term,term): stat;
  >(term,term): stat;
  <(term,term): stat;
  in(term,set): stat;
  notin(term,set): stat;
  is(term,noun): stat;
  sq(term): term;
  sqrt(term): term;
  *(term,term): term;
  +(term,term): term;
  /(term,term): term;
  abs(term): term;
  subtraction(set,set): set;
  one_element_set(term): set;
  number: noun;
  even: adj;
  sequence(set):noun;
  infinite:adj;
  descending:adj; };

 Lemma := forall( m:N, forall(n:N,
           implies( =(sq(m),*(2,sq(n))) , and(=(m,n), =(n,0)) )));
 {
   { m:N; n:N;} |>
   {
    { m1:N; } |>
      P(m1):= exists(n1:N, and(=(sq(m1),*(2,sq(n1))),>(m1,0)) );
    Claim := implies(P(m), exists(m':N, and(<(m',m),P(m'))) );
    {
     { =(sq(m),*(2,sq(n))); >(m,0); } |>
     {
       is(sq(m), even number);
       is(m, even number);
       { k:N; =(m,*(2,k)); } |>
       {
         implies(and( =( *(2,sq(n)),sq(m)), =(sq(m),*(4,sq(k)))),
                  =(sq(n),*(2,sq(k))) );
       };
       { >(sq(m),0); } |>
       {
         >(sq(n),0);
         >(n,0);
```

```
      };
      P(n);
      =(sq(m),+(sq(n),sq(n)));
      >(+(sq(n),sq(n)),sq(n));
      >(sq(m),sq(n));
      >(m,n);
      { m':N; } |> =(m',n) ;
    };
   };
   { Claim;
     not( exists(s:sequence(N),
             and(is(s,infinite sequence(N)),is(s,descending  sequence(N))) ));
   } |> forall(m1:N, not(P(m1)) );
   { =(sq(m),*(2,sq(n))); } |>
   {
     { neq(m,0); } |>
     { >(m,0);
       P(m);
       contradiction;
     };
     {=(m,0);} |> =(n,0);
   };
  };
 };


Corollary := notin(sqrt(2),Q);
{
 { in(sqrt(2),Q); } |>
 {
  p:Z;
  q:subtraction(Z,one_element_set(0));
  =(sqrt(2),/(p,q));
  { m:N; =(m,abs(p)); n:N; =(n,abs(q)); neq(n,0); =(sqrt(2),/(m,n));} |>
  {
    =(sq(m),*(2,sq(n)));
    Lemma |>  =(n,0);
    contradiction;
  };
 };
};
```

Listing A.5: The MathLang CGa *plain syntax* version of the proof of Pythagoras' theorem by H. Barendregt.

```
:: 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;
   suppose B1: m <> 0;
     then m > 0;
::>           *4
     then P[m] by B1;
::>              *4
     then contradiction by A2;
     hence thesis;
   end;
   suppose S1: m = 0;
     then n = 0;
::>           *4
     thus thesis by S1;
::>               *4
   end;
 end;

Corollary:  sqrt 2 is irrational
  proof
    assume sqrt 2 is rational;
    then ex p,q being Integer st
    q <> 0 & sqrt 2 = p/q;
::>                      *4
    then consider m,n being Integer such that
    A0: sqrt 2 = m/n and m = abs m & n = abs n & n <> 0;
::>                                                  *4
    m^2 = 2*n^2;
::>           *4
    n = 0 by Lemma;
::>        *4
    hence contradiction;
::>                   *4
  end;

::> 4: This inference is not accepted
```

Listing A.6: The Mizar FPS version of the proof of Pythagoras' theorem by H. Barendregt.

```
:: 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 SQUARE_1, IRRAT_1, RAT_1, MATRIX_2, INT_1, ARYTM_3, ABSVALUE,
    FINSET_1, SEQM_3, XREAL_0, ARYTM;
 notations INT_1, SQUARE_1, IRRAT_1, XCMPLX_0, RAT_1, REAL_1, ABIAN,
    PEPIN, INT_2, FINSET_1, SEQM_3, XXREAL_0, XREAL_0, NAT_1, SUBSET_1,
    NUMBERS, ORDINAL1;
 constructors NAT_1, SQUARE_1, IRRAT_1, XCMPLX_0, RAT_1, ABIAN, INT_1,
    PEPIN, INT_2, FINSET_1, SEQM_3, XXREAL_0, XREAL_0,SUBSET_1;
 requirements SUBSET, NUMERALS, ARITHM, BOOLE, REAL;
 registrations XREAL_0, REAL_1, NAT_1, INT_1;
 theorems ABIAN,PYTHTRIP,SQUARE_1,XCMPLX_1,REAL_1,NAT_1,INT_1,
    RAT_1,REAL_2,XREAL_1, ORDINAL1;
 schemes NAT_1;
begin

Lemma:
 for m,n being Nat holds m^2 = 2*n^2 implies m = 0 & n = 0
 proof
   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 B0: P[m];
     then consider n being Nat such that
     B1: m^2 = 2*n^2 &  m > 0;
     m^2 is even by ABIAN:def 1,B1;
     then m is even by PYTHTRIP:2;
     then consider k being Integer such that
     B2: m = 2*k by ABIAN:def 1;
     2*n^2 = m^2 by B1
          .= 2^2*k^2 by B2,SQUARE_1:68
          .= (2*2)*k^2 by SQUARE_1:def 3
          .= 4*k^2;
     then B3: n^2 = 2*k^2;
     m^2 > 0 by SQUARE_1:74,B0;
     then (2*n^2)/2 > 0/2 by B1,REAL_1:73;
     then B6: n^2 > 0;
     then n <> 0 by SQUARE_1:60,B1;
     then B8: n > 0 by NAT_1:3;
     0 <= k
     proof
       assume not thesis;
       then k < 0;
       then 2*k < 2*0 by XREAL_1:70;
       hence contradiction by B2,B1;
     end;
     then k is Element of NAT by INT_1:16;
     then reconsider k as Nat;
```

```
    C0: n^2 = 2*k^2 & n > 0 by B3,B8;
    then C1: P[n];
    0+n^2 < n^2 + n^2 by B6,XREAL_1:10;
    then m^2 = n^2 + n^2 & n^2 + n^2 > n^2 by B1;
    then m^2 > n^2 ;
    then sqrt m^2 > sqrt n^2 by B6,SQUARE_1:95;
    then 0 <= m & sqrt m^2 > n by B1,C0,SQUARE_1:89;
    then C2: m > n by SQUARE_1:89;
    take m' = n;
    thus thesis by C1,C2;
  end;
  A2: for k being Nat holds not P[k]
  proof
    let k be Nat;
    assume S0: not thesis;
    then S1: ex k being Nat st P[k];
    then consider aa being Nat such that B1: P[aa];
    reconsider aa as Element of NAT by ORDINAL1:def 13;
    P[aa] by B1;
    then S11: ex k being Element of NAT st P[k];
    S2: for k being Nat st k <> 0 & P[k] holds
    ex n being Nat st n < k & P[n] by Claim;
    S22: for k being Element of NAT st k <> 0 & P[k] holds
    ex n being Element of NAT st n < k & P[n]
    proof
      let k be Element of NAT;
      assume that T0: k <> 0 and T1: P[k];
      ex n being Nat st n < k & P[n] by T0,T1,S2;
      consider n being Nat such that
      T2: n < k & P[n] by T0,T1,S2;
      take n;
      reconsider n as Element of NAT by ORDINAL1:def 13;
      n < k & P[n] by T2;
      hence thesis;
    end;
    P[0] from NAT_1:sch 7(S11,S22);
    hence contradiction;
  end;
  let m,n being Nat;
  assume A0: m^2 = 2*n^2;
  per cases by A0;
  suppose B1: m^2 = 2*n^2 & m <> 0;
    then m > 0 by NAT_1:3;
    then P[m] by B1;
    then contradiction by A2;
    hence thesis;
  end;
  suppose S1: m^2 = 2*n^2 & m = 0;
    then 0 = 2*n^2 by SQUARE_1:60;
    then 0 = n*n by SQUARE_1:def 3;
    then 0 = n by XCMPLX_1:6;
    hence thesis by S1;
  end;
end;
```

```
Corollary1:
  sqrt 2 is irrational
  proof
    assume sqrt 2 is rational;
    then consider m being Integer, n being Element of NAT such that
    A0: n <> 0 & sqrt 2 = m/n by RAT_1:24;
    A1: m = sqrt 2 * n by A0,XCMPLX_1:88;
    A2: m/n >= 0 by A0, SQUARE_1:def 4;
    then n >= 0 by NAT_1:3;
    then m >= 0 by A2,A1,REAL_2:121,A0;
    then m is Element of NAT by INT_1:16;
    then reconsider m as Nat;
    reconsider n as Nat;
    m^2 = (sqrt 2)^2*n^2 by A1,SQUARE_1:68
       .=2*n^2 by SQUARE_1:def 4;
    then m = 0 & n = 0 by Lemma;
    hence contradiction by A0;
  end;
```

Listing A.7: The full formalisation in Mizar of the proof of Pythagoras' theorem by H. Barendregt.

A.4 The DRa example of annotating "Foundations of Analysis" by E. Landau

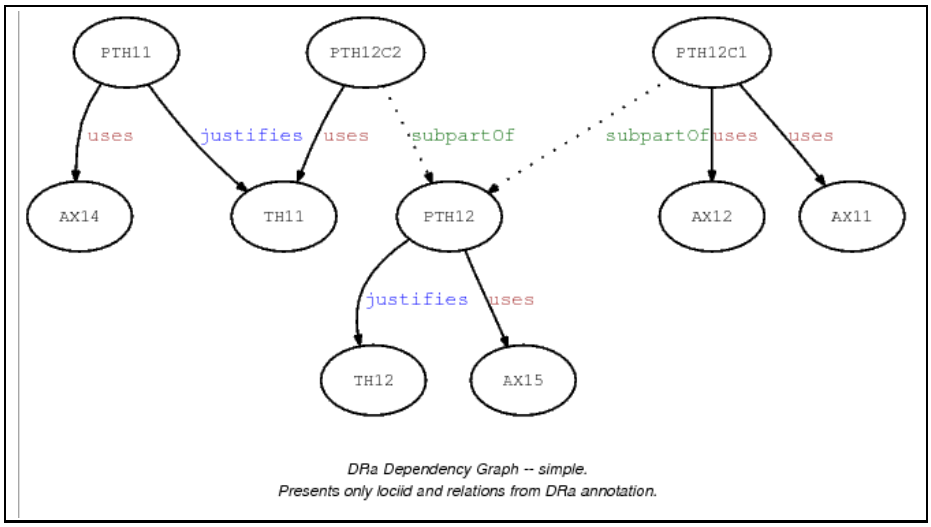


Figure A.7: A TEXMACS generated *DG* for the example from Figure A.9.

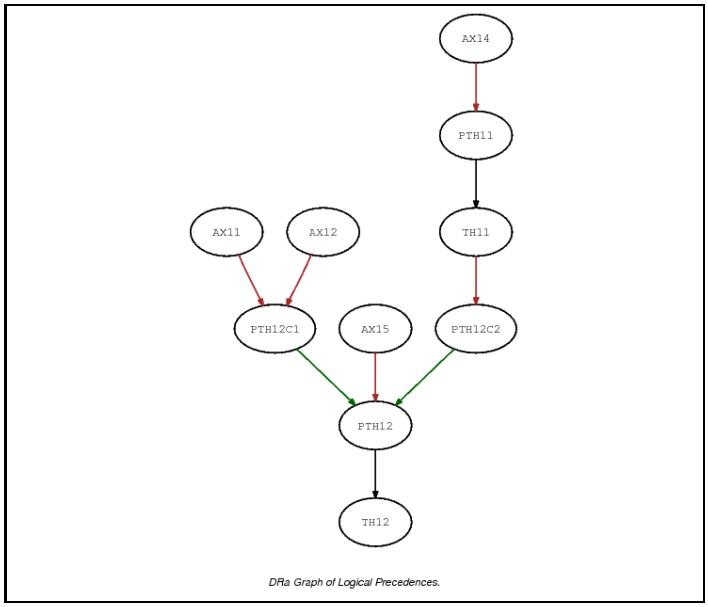


Figure A.8: A TEXMACS generated *GoLP* for the example from Figure A.9.

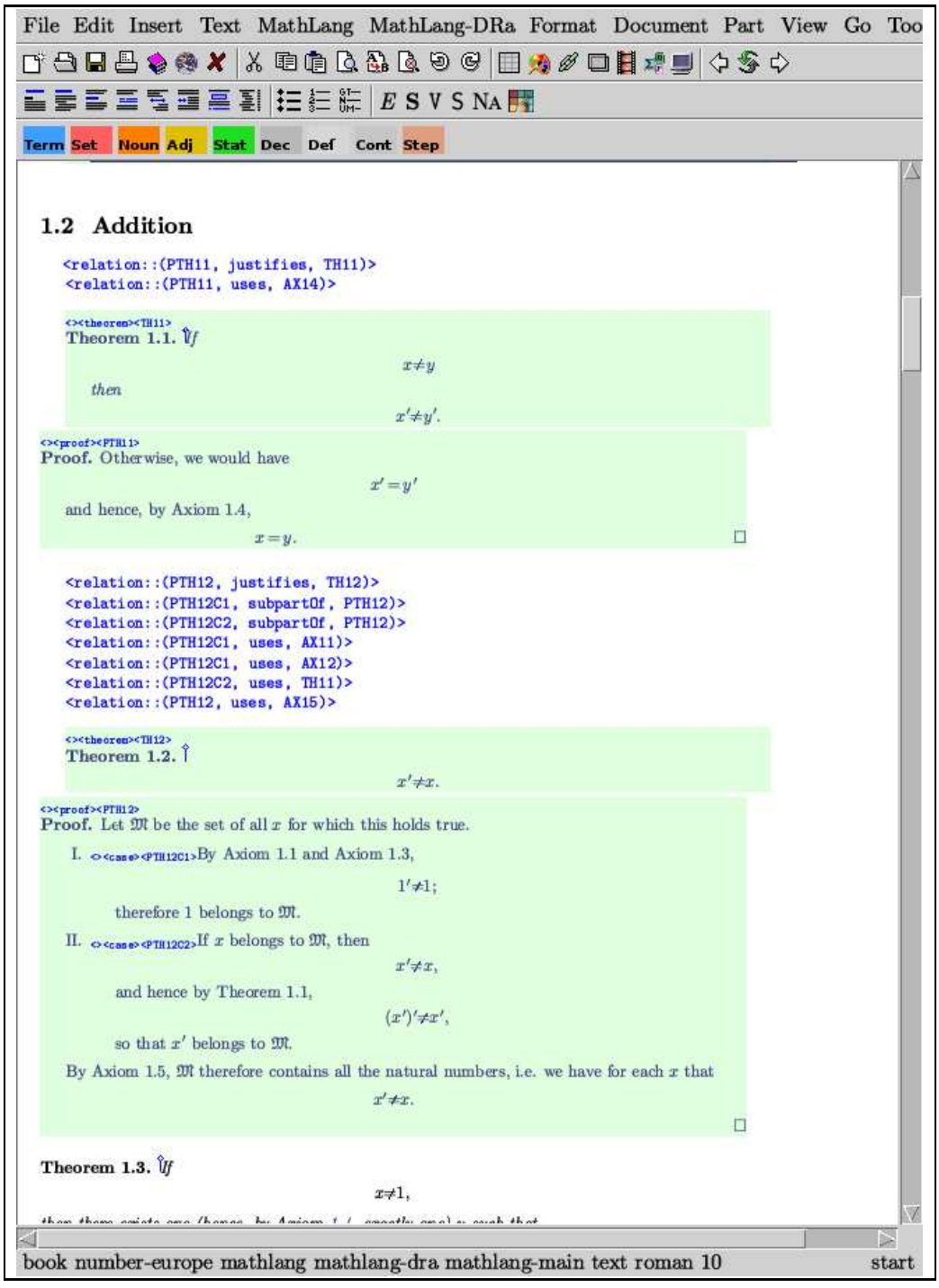


Figure A.9: A fragment of the MathLang DRa annotation of the first chapter of "Foundations of Analysis" E. Landau [Lan51].

Appendix B

Transformation functions and stylesheets for the DRa

B.1 XSLT stylesheet

```
<xsl:template match="dra:relation">
   <xsl:call-template name="arrow">
     <xsl:with-param name="id" select="@xml:id" />
     <xsl:with-param name="type" select="dra:type"/>
     <xsl:with-param name="arg1" select="dra:source"/>
     <xsl:with-param name="arg2" select="dra:target"/>
   </xsl:call-template>
 </xsl:template>

 <xsl:template name="arrow">
   <xsl:param name="id"/>
   <xsl:param name="type"/>
   <xsl:param name="arg1"/>
   <xsl:param name="arg2"/>
   <xsl:value-of select="$space"/>
   <xsl:value-of select="$arg1"/>
   <xsl:text> -> </xsl:text>
   <xsl:value-of select="$arg2"/>
   <xsl:text> [style=</xsl:text>
   <xsl:choose>
     <xsl:when test="$type = 'justifies'">
       <xsl:text>solid</xsl:text>
     </xsl:when>
     <xsl:when test="$type = 'uses'">
       <xsl:text>dashed</xsl:text>
     </xsl:when>
     <xsl:when test="$type = 'subpartOf'">
       <xsl:text>dotted</xsl:text>
     </xsl:when>
     <xsl:when test="$type = 'inconsistentWith'">
       <xsl:text>dotted</xsl:text>
     </xsl:when>
     <xsl:when test="$type = 'exemplifies'">
       <xsl:text>dotted</xsl:text>
     </xsl:when>
     <xsl:when test="$type = 'relatesTo'">
       <xsl:text>dotted</xsl:text>
     </xsl:when>
     <xsl:otherwise>
       <xsl:text>bold</xsl:text>
     </xsl:otherwise>
   </xsl:choose>
   <xsl:text>, label="</xsl:text>
   <xsl:value-of select="$type"/>
   <xsl:text>-</xsl:text>
   <xsl:value-of select="$id"/>
   <xsl:text>"]; </xsl:text>
   <xsl:value-of select="$newline"/>
 </xsl:template>
```

Figure B.1: The fragment of the XSLT file: `mathlang-dra2dg.xsl`. It presents XSL templates built to retrieve DRa relation annotations from the MathLang-DRa annotated document. It also presents templates that transform the relations into their descriptive presentation in a text file using the DOT language.

B.2 SCHEME implementation of the transformation stylesheet

```
;; MathLang DRa dependency graph - simple, with LociId.
(tm-define (mathlang-tree-dg-A t mf) ; t -- tree, mf -- mathlang-file
  (if (tree? t)
      (if (tree-atomic? t)
          (if (== (tree-arity t) 0)
              (if (and (list? (tree->stree t)) (eq (cAr (tree->stree t)) 'concat))
                  (foreach x (tree->stree t)
                           (mathlang-tree-dg-A x mf))
                  )
          (let* ((n (tree-arity t))
                 (l (tree-label t))
                 )
            (cond
             ((== l 'MathLang-DRa) ;; <dra:node> tag
              (let* ((LociId (tree->stree (tree-ref t 2))))
                (if (not (equal? LociId ""))
                    (begin
                      (mathlang-socket-print mf LociId)
                      (mathlang-socket-print mf " [ label= ")
                      (mathlang-socket-print mf LociId)
                      (mathlang-socket-print-newline mf " ];")
                      )
                    (begin
                      (display "FIXME_Lociid\n")
                      (mathlang-socket-print-newline mf " [ label=FIXME_Lociid ];")
                      ))
                )
              (foreach-number (i 0 < (tree-arity t))
                           (mathlang-tree-dg-A (tree-ref t i) mf))
              )
             ((== l 'MathLang-DRa-relation) ;; <dra:relation> tag
              (mathlang-socket-print mf (tree->stree (tree-ref t 2)))
              (mathlang-socket-print mf " -> ")
              (mathlang-socket-print mf (tree->stree (tree-ref t 3)))
              (mathlang-socket-print mf " [style=")
              (if (equal? (tree->stree (tree-ref t 0)) "justifies")
                  (mathlang-socket-print mf "solid, fontcolor=blue,"))
              (if (equal? (tree->stree (tree-ref t 0)) "uses")
                  (mathlang-socket-print mf "solid, fontcolor=brown,"))
              (if (equal? (tree->stree (tree-ref t 0)) "subpartOf")
                  (mathlang-socket-print mf "dotted, fontcolor=darkgreen,"))
              (if (equal? (tree->stree (tree-ref t 0)) "inconsistentWith")
                  (mathlang-socket-print mf "solid, fontcolor=brown,"))
              (if (equal? (tree->stree (tree-ref t 0)) "exemplifies")
                  (mathlang-socket-print mf "solid, fontcolor=brown,"))
              (if (equal? (tree->stree (tree-ref t 0)) "relatesTo")
                  (mathlang-socket-print mf "solid, fontcolor=darkorange4,"))
              (if (and (not (equal? (tree->stree (tree-ref t 0)) "justifies"))
                       (not (equal? (tree->stree (tree-ref t 0)) "uses"))
                       (not (equal? (tree->stree (tree-ref t 0)) "subpartOf"))
                       (not (equal? (tree->stree (tree-ref t 0)) "inconsistentWith"))
                       (not (equal? (tree->stree (tree-ref t 0)) "exemplifies"))
                       (not (equal? (tree->stree (tree-ref t 0)) "relatesTo"))
                       )
                  (mathlang-socket-print mf "bold, fontcolor=red,"))
              (mathlang-socket-print mf " label=")
              (if (equal? (tree->stree (tree-ref t 0)) "")
                  (begin
                    (display "Something wrong!!!.\n")
                    (mathlang-socket-print-newline mf "FIXME_type") )
                  (mathlang-socket-print mf (tree->stree (tree-ref t 0)))
                  )
              (mathlang-socket-print-newline mf "];")
              (foreach-number (i 0 < (tree-arity t))
                           (mathlang-tree-dg-A (tree-ref t i) mf))
              )
```

Figure B.2: The presentation fragment of the SCHEME implementation responsible for generating *Dependency Graph* directly from the TEXMACS editor. This function is associated with one of the MathLang-DRa TEXMACS plugin functions. It is equivalent to the XSLT stylesheet presented in Figure B.1.

Appendix C

Mizar formalisation attempts performed by the student

C.1 Formalisation of series-parallel graphs

```
:: The Class of Series --- Parallel Graphs, {I}
::  by Krzysztof Retel
::
:: Received November 18, 2002
:: Copyright (c) 2002 Association of Mizar Users

environ

 vocabularies ORDINAL2, ARYTM_3, NECKLACE, FUNCT_3, ARYTM, FUNCT_2, ORDERS_1,
      RELAT_1, RELAT_2, REALSET1, FUNCT_1, BOOLE, SEQM_3, SUBSET_1, WELLORD1,
      CAT_1, FUNCOP_1, FUNCT_4, ARYTM_1, TARSKI, CARD_1, CARD_2, FINSET_1,
      SQUARE_1;
 notations TARSKI, XBOOLE_0, ENUMSET1, SUBSET_1, ZFMISC_1, CARD_1, CARD_2,
      WELLORD1, FINSET_1, SQUARE_1, QUIN_1, REALSET1, ORDINAL1, NUMBERS,
      XCMPLX_0, REAL_1, NAT_1, NAT_D, RELAT_1, RELAT_2, BINARITH, FUNCT_3,
      FUNCOP_1, RELSET_1, FUNCT_1, PARTFUN1, FUNCT_2, FUNCT_4, BINOP_1,
      XXREAL_0, STRUCT_0, ORDERS_2, WAYBEL_0, WAYBEL_1, ORDERS_3;
 constructors WELLORD1, REAL_1, SQUARE_1, NAT_1, QUIN_1, CARD_2, REALSET1,
      BINARITH, ORDERS_3, WAYBEL_1, NAT_D;
 registrations XBOOLE_0, SUBSET_1, FUNCT_1, ORDINAL1, RELSET_1, FUNCT_2,
      FUNCOP_1, FUNCT_4, FINSET_1, XXREAL_0, XREAL_0, SQUARE_1, REALSET1,
      STRUCT_0, ORDERS_2, ORDERS_3, HILBERT3, CARD_1;
 requirements BOOLE, SUBSET, NUMERALS, REAL, ARITHM;


begin :: Preliminaries

reserve i,j,k,n for Nat;
reserve x,x1,x2,x3,y1,y2,y3 for set;

canceled;
```

```
theorem :: NECKLACE:2
  4 = {0,1,2,3};

theorem :: NECKLACE:3
  [:{x1,x2,x3},{y1,y2,y3}:] =
  {[x1,y1],[x1,y2],[x1,y3],[x2,y1],[x2,y2],[x2,y3],[x3,y1],[x3,y2],[x3,y3]};

theorem :: NECKLACE:4
  for x being set, n be Nat holds x in n implies x is Nat;

theorem :: NECKLACE:5
  for x be non empty Nat holds 0 in x;

registration
  let X be set;
  cluster delta X -> one-to-one;
end;

theorem :: NECKLACE:6
  for X being set holds Card id X = Card X;

registration
  let R be trivial Relation;
  cluster dom R -> trivial;
end;

registration
  cluster trivial -> one-to-one Function;
end;

theorem :: NECKLACE:7
  for f,g be Function st dom f misses dom g holds
  rng(f +* g) = rng f \/ rng g;

theorem :: NECKLACE:8
  for f,g be one-to-one Function st dom f misses dom g &
  rng f misses rng g holds (f+*g)" = f" +* g";

theorem :: NECKLACE:9
  for A,a,b being set holds (A --> a) +* (A --> b) = A --> b;

theorem :: NECKLACE:10
  for a,b being set holds (a .--> b)" = b .--> a;

theorem :: NECKLACE:11
  for a,b,c,d being set st a = b iff c = d
  holds (a,b) --> (c,d)" = (c,d) --> (a,b);

scheme :: NECKLACE:sch 1
 Convers{X()-> non empty set, R() -> Relation, F,G(set)-> set, P[set]}:
  R()~ ={[F(x),G(x)] where x is Element of X(): P[x]}
provided
 R() = {[G(x),F(x)] where x is Element of X(): P[x]};
```

```
theorem :: NECKLACE:12
  for i,j,n be Nat holds i < j & j in n implies i in n;

begin   :: Auxiliary Concepts

definition
  let R,S be RelStr;
  canceled;
  pred S embeds R means
:: NECKLACE:def 2

  ex f being Function of R,S st f is one-to-one
  & for x,y being Element of R holds
  [x,y] in the InternalRel of R iff [f.x,f.y] in the InternalRel of S;
end;

definition
  let R,S be non empty RelStr;
  redefine pred S embeds R;
  reflexivity;
end;

theorem :: NECKLACE:13
  for R,S,T be non empty RelStr holds
  R embeds S & S embeds T implies R embeds T;

definition
  let R,S be non empty RelStr;
  pred R is_equimorphic_to S means
:: NECKLACE:def 3

  R embeds S & S embeds R;
  reflexivity;
  symmetry;
end;

theorem :: NECKLACE:14
  for R,S,T be non empty RelStr holds
  R is_equimorphic_to S & S is_equimorphic_to T implies R is_equimorphic_to T;

notation
  let R be non empty RelStr;
  antonym R is parallel for R is connected;
end;

definition
  let R be RelStr;
  attr R is symmetric means
:: NECKLACE:def 4

  the InternalRel of R is_symmetric_in the carrier of R;
end;

definition
  let R be RelStr;
```

```
  attr R is asymmetric means
:: NECKLACE:def 5

  the InternalRel of R is asymmetric;
end;

theorem :: NECKLACE:15
  for R be RelStr st R is asymmetric holds
  the InternalRel of R misses (the InternalRel of R)~;

definition
  let R be RelStr;
  attr R is irreflexive means
:: NECKLACE:def 6

  for x being set st x in the carrier of R
  holds not [x,x] in the InternalRel of R;
end;

definition
  let n be Nat;
  func n-SuccRelStr -> strict RelStr means
:: NECKLACE:def 7

  the carrier of it = n &
  the InternalRel of it = {[i,i+1] where i is Element of NAT:i+1 < n};
end;

theorem :: NECKLACE:16
  for n be Nat holds n-SuccRelStr is asymmetric;

theorem :: NECKLACE:17
  n > 0 implies Card the InternalRel of n-SuccRelStr = n-1;

definition
  let R be RelStr;
  func SymRelStr R -> strict RelStr means
:: NECKLACE:def 8

  the carrier of it = the carrier of R &
  the InternalRel of it = (the InternalRel of R) \/ (the InternalRel of R)~;
end;

registration
  let R be RelStr;
  cluster SymRelStr R -> symmetric;
end;

registration
  cluster non empty symmetric RelStr;
end;

registration
  let R be symmetric RelStr;
  cluster the InternalRel of R -> symmetric;
```

```
end;

definition
  let R be RelStr;
  func ComplRelStr R -> strict RelStr means
:: NECKLACE:def 9

  the carrier of it = the carrier of R &
  the InternalRel of it = (the InternalRel of R)` \ id (the carrier of R);
end;

registration
  let R be non empty RelStr;
  cluster ComplRelStr R -> non empty;
end;

theorem :: NECKLACE:18
  for S,R being RelStr holds S,R are_isomorphic implies
  Card the InternalRel of S = Card the InternalRel of R;

begin   :: Necklace n

definition
  let n be Nat;
  func Necklace n -> strict RelStr equals
:: NECKLACE:def 10

  SymRelStr(n-SuccRelStr);
end;

registration
  let n be Nat;
  cluster Necklace n -> symmetric;
end;

theorem :: NECKLACE:19
  the InternalRel of Necklace n
  = {[i,i+1] where i is Element of NAT:i+1 < n} \/
  {[i+1,i] where i is Element of NAT:i+1 < n};

theorem :: NECKLACE:20
  for x be set holds x in the InternalRel of Necklace n iff
  ex i being Element of NAT st i+1 < n & (x = [i,i+1] or x = [i+1,i]);

registration
  let n be Nat;
  cluster Necklace n -> irreflexive;
end;

theorem :: NECKLACE:21
  for n be Nat holds the carrier of Necklace n = n;

registration
  let n be non empty Nat;
  cluster Necklace n -> non empty;
```

```
end;

registration
  let n be Nat;
  cluster the carrier of Necklace n -> finite;
end;

theorem :: NECKLACE:22
  for n,i be Nat st i+1 < n holds [i,i+1] in the InternalRel of Necklace n;

theorem :: NECKLACE:23
  for n be Nat, i being Nat st i in the carrier of Necklace n holds i < n;

theorem :: NECKLACE:24
  for n be non empty Nat holds Necklace n is connected;

theorem :: NECKLACE:25
  for i,j being Nat st [i,j] in the InternalRel of Necklace n
  holds i = j + 1 or j = i + 1;

theorem :: NECKLACE:26
  for i,j being Nat st (i = j + 1 or j = i + 1) &
  i in the carrier of Necklace n & j in the carrier of Necklace n
  holds [i,j] in the InternalRel of Necklace n;

theorem :: NECKLACE:27
  n > 0 implies Card ({[i+1,i] where i is Element of NAT:i+1 < n}) = n-1;

theorem :: NECKLACE:28
  n > 0 implies Card the InternalRel of Necklace n = 2*(n-1);

theorem :: NECKLACE:29
  Necklace 1, ComplRelStr Necklace 1 are_isomorphic;

theorem :: NECKLACE:30
  Necklace 4, ComplRelStr Necklace 4 are_isomorphic;

theorem :: NECKLACE:31
  Necklace n, ComplRelStr Necklace n are_isomorphic implies
  n = 0 or n = 1 or n = 4;
```

Listing C.1: The Mizar article `NECKLACE.abs`, the first article from the series, formalising "series-parallel" graphs of the original article [Tho00].

```
:: The Class of Series-Parallel Graphs, {II}
::  by Krzysztof Retel
::
:: Received May 29, 2003
:: Copyright (c) 2003 Association of Mizar Users

environ

 vocabularies NECKLA_2, NECKLACE, CLASSES1, CLASSES2, ORDERS_1, RELAT_1,
      REALSET1, FUNCT_1, BOOLE, SETFAM_1, CANTOR_1, CARD_1, PROB_1, SQUARE_1,
      ARYTM, FINSET_1, ORDINAL2, ORDINAL1;
 notations TARSKI, XBOOLE_0, ENUMSET1, SUBSET_1, ZFMISC_1, RELSET_1, FINSET_1,
      CARD_1, ORDINAL1, NUMBERS, NAT_1, REALSET1, RELAT_1, FUNCT_1, FUNCT_2,
      STRUCT_0, ORDERS_2, NECKLACE, CLASSES2, SETFAM_1, CLASSES1, CARD_3,
      XXREAL_0;
 constructors SQUARE_1, NAT_1, CARD_3, CLASSES2, REALSET2, COH_SP, NECKLACE;
 registrations XBOOLE_0, SUBSET_1, ORDINAL1, RELSET_1, FINSET_1, XREAL_0,
      NAT_1, CLASSES2, STRUCT_0, ORDERS_2, YELLOW13, CARD_1;
 requirements BOOLE, SUBSET, REAL, NUMERALS, ARITHM;


begin

reserve U for Universe;

theorem :: NECKLA_2:1
  for X,Y being set st X in U & Y in U
  for R being Relation of X,Y holds R in U;

theorem :: NECKLA_2:2
  the InternalRel of Necklace 4 = {[0,1],[1,0],[1,2],[2,1],[2,3],[3,2]};

registration
  let n be natural number;
  cluster -> finite Element of Rank n;
end;

theorem :: NECKLA_2:3
  for x be set st x in FinSETS holds x is finite;

registration
  cluster -> finite Element of FinSETS;
end;

definition
  let G be non empty RelStr;
  attr G is N-free means
:: NECKLA_2:def 1

  not G embeds Necklace 4;
end;

registration
  cluster strict finite N-free (non empty RelStr);
```

```
end;

definition
  let R,S be RelStr;
  func union_of(R,S) -> strict RelStr means
:: NECKLA_2:def 2

  the carrier of it = (the carrier of R) \/ (the carrier of S) &
  the InternalRel of it = (the InternalRel of R) \/ (the InternalRel of S);
end;

definition
  let R, S be RelStr;
  func sum_of(R,S) -> strict RelStr means
:: NECKLA_2:def 3

  the carrier of it = (the carrier of R) \/ (the carrier of S) &
  the InternalRel of it = (the InternalRel of R) \/ (the InternalRel of S)
  \/ [:the carrier of R, the carrier of S:]
  \/ [:the carrier of S, the carrier of R:];
end;

definition
  func fin_RelStr means
:: NECKLA_2:def 4

  for X being set holds X in it iff
  ex R being strict RelStr st X = R & the carrier of R in FinSETS;
end;

registration
  cluster fin_RelStr -> non empty;
end;

definition
  func fin_RelStr_sp -> Subset of fin_RelStr means
:: NECKLA_2:def 5

  (for R be strict RelStr st the carrier of R is non empty trivial
  & the carrier of R in FinSETS holds R in it) &
  (for H1,H2 be strict RelStr st
  (the carrier of H1) misses (the carrier of H2) & H1 in it & H2 in it
  holds union_of(H1,H2) in it & sum_of(H1,H2) in it) &
  for M be Subset of fin_RelStr st
  ( (for R be strict RelStr st the carrier of R is non empty trivial &
  the carrier of R in FinSETS holds R in M) & for H1,H2 be strict RelStr st
  (the carrier of H1) misses (the carrier of H2) & H1 in M & H2 in M
  holds union_of(H1,H2) in M & sum_of(H1,H2) in M ) holds it c= M;
end;

registration
  cluster fin_RelStr_sp -> non empty;
end;

theorem :: NECKLA_2:4
```

```
  for X being set st X in fin_RelStr_sp holds
  X is finite strict non empty RelStr;

theorem :: NECKLA_2:5
  for R being RelStr st R in fin_RelStr_sp holds (the carrier of R) in FinSETS;

theorem :: NECKLA_2:6
  for X being set st X in fin_RelStr_sp holds
  X is strict non empty trivial RelStr or ex H1,H2 being strict RelStr st
  (the carrier of H1) misses (the carrier of H2) &
  H1 in fin_RelStr_sp & H2 in fin_RelStr_sp &
  (X = union_of(H1,H2) or X = sum_of(H1,H2) );

theorem :: NECKLA_2:7
  for R being strict non empty RelStr st R in fin_RelStr_sp holds R is N-free;
```

Listing C.2: The Mizar article `NECKLACE_2.abs`, the second article from the series, formalising "series-parallel" graphs of the original article [Tho00].

```
:: The Class of Series-Parallel Graphs, {III}
::  by Krzysztof Retel
::
:: Received February 3, 2004
:: Copyright (c) 2004 Association of Mizar Users

environ

 vocabularies NECKLA_3, NECKLA_2, NECKLACE, ORDERS_1, RELAT_1, FINSET_1,
      FUNCT_1, BOOLE, YELLOW_0, SUBSET_1, INCPROJ, RELAT_2, CLASSES2, CAT_1,
      REALSET1, REWRITE1, CARD_1, FINSEQ_1, ARYTM_1, FINSEQ_5, MSUALG_5,
      TARSKI, EQREL_1, WELLORD1, FUNCT_4, FUNCOP_1, SEQM_3, ARYTM;
 notations TARSKI, XBOOLE_0, ENUMSET1, SUBSET_1, ZFMISC_1, RELAT_1, RELAT_2,
      RELSET_1, FUNCT_1, PARTFUN1, FUNCT_2, FINSET_1, CARD_1, NUMBERS,
      XCMPLX_0, XXREAL_0, NAT_1, REALSET1, DOMAIN_1, STRUCT_0, ORDERS_2,
      WAYBEL_1, FUNCOP_1, CLASSES2, NECKLACE, WAYBEL_0, FUNCT_4, REWRITE1,
      FINSEQ_1, FINSEQ_5, EQREL_1, MSUALG_5, YELLOW_0, NECKLA_2, WELLORD1;
 constructors XXREAL_0, NAT_1, BINOP_2, EQREL_1, CLASSES1, TOLER_1, CLASSES2,
      FINSEQ_5, REWRITE1, REALSET2, MSUALG_5, ORDERS_3, WAYBEL_1, NECKLACE,
      NECKLA_2;
 registrations XBOOLE_0, SUBSET_1, RELAT_1, FUNCT_1, PARTFUN1, FUNCT_2,
      FUNCOP_1, FINSET_1, XXREAL_0, XREAL_0, NAT_1, CARD_1, FINSEQ_1, REALSET1,
      STRUCT_0, ORDERS_2, YELLOW_0, ORDERS_4, NECKLACE, NECKLA_2, ORDINAL1,
      FUNCT_4;
 requirements BOOLE, SUBSET, REAL, NUMERALS, ARITHM;


begin :: Preliminaries

reserve A,B,a,b,c,d,e,f,g,h for set;

theorem :: NECKLA_3:1
  (id A)|B = id A /\ [:B,B:];

theorem :: NECKLA_3:2
  id {a,b,c,d} = {[a,a],[b,b],[c,c],[d,d]};

theorem :: NECKLA_3:3
  [:{a,b,c,d},{e,f,g,h}:] = {[a,e],[a,f],[b,e],[b,f],[a,g],[a,h],[b,g],[b,h]}
  \/ {[c,e],[c,f],[d,e],[d,f],[c,g],[c,h],[d,g],[d,h]};

registration
  let X,Y be trivial set;
  cluster -> trivial Relation of X,Y;
end;

theorem :: NECKLA_3:4
  for X be trivial set, R be Relation of X st R is non empty
  holds ex x be set st R = {[x,x]};

registration
  let X be trivial set;
  cluster -> trivial reflexive symmetric transitive
    strongly_connected Relation of X;
```

```
end;

theorem :: NECKLA_3:5
  for X be non empty trivial set, R be Relation of X holds R is_symmetric_in X;

registration
  cluster non empty strict finite irreflexive symmetric RelStr;
end;

registration
  let L be irreflexive RelStr;
  cluster -> irreflexive (full SubRelStr of L);
end;

registration
  let L be symmetric RelStr;
  cluster -> symmetric (full SubRelStr of L);
end;

theorem :: NECKLA_3:6
  for R be irreflexive symmetric RelStr st Card (the carrier of R) = 2
  holds ex a,b be set st the carrier of R = {a,b} &
  (the InternalRel of R = {[a,b],[b,a]} or the InternalRel of R = {});

begin :: Some facts about operations 'union_of' and 'sum_of'

registration
  let R be non empty RelStr,S be RelStr;
  cluster union_of(R,S) -> non empty;
  cluster sum_of(R,S) -> non empty;
end;

registration
  let R be RelStr, S be non empty RelStr;
  cluster union_of(R,S) -> non empty;
  cluster sum_of(R,S) -> non empty;
end;

registration
  let R,S be finite RelStr;
  cluster union_of(R,S) -> finite;
  cluster sum_of(R,S) -> finite;
end;

registration
  let R,S be symmetric RelStr;
  cluster union_of(R,S) -> symmetric;
  cluster sum_of(R,S) -> symmetric;
end;

registration
  let R,S be irreflexive RelStr;
  cluster union_of(R,S) -> irreflexive;
end;
```

```
theorem :: NECKLA_3:7
  for R,S be irreflexive RelStr st the carrier of R misses the carrier of S
  holds sum_of(R,S) is irreflexive;

theorem :: NECKLA_3:8
  for R1,R2 being RelStr holds
  union_of(R1,R2) = union_of(R2,R1) & sum_of(R1,R2) = sum_of(R2,R1);

theorem :: NECKLA_3:9
  for G being irreflexive RelStr, G1,G2 being RelStr st
  ( G = union_of(G1,G2) or G = sum_of(G1,G2) )
  holds G1 is irreflexive & G2 is irreflexive;

theorem :: NECKLA_3:10
  for G being non empty RelStr, H1,H2 being RelStr
  st the carrier of H1 misses the carrier of H2 &
  ( the RelStr of G = union_of(H1,H2) or the RelStr of G = sum_of(H1,H2) )
  holds H1 is full SubRelStr of G & H2 is full SubRelStr of G;

begin :: Theorems relating to the complement of RelStr

theorem :: NECKLA_3:11
  the InternalRel of ComplRelStr Necklace 4
  = {[0,2],[2,0],[0,3],[3,0],[1,3],[3,1]};

registration
  let R be RelStr;
  cluster ComplRelStr R -> irreflexive;
end;

registration
  let R be symmetric RelStr;
  cluster ComplRelStr R -> symmetric;
end;

theorem :: NECKLA_3:12
  for R be RelStr holds
  the InternalRel of R misses the InternalRel of ComplRelStr R;

theorem :: NECKLA_3:13
  for R being RelStr holds
  id the carrier of R misses the InternalRel of ComplRelStr R;

theorem :: NECKLA_3:14
  for G being RelStr holds
  [:the carrier of G,the carrier of G:] = id (the carrier of G) \/
  (the InternalRel of G) \/ (the InternalRel of ComplRelStr G);

theorem :: NECKLA_3:15
  for G being strict irreflexive RelStr st G is trivial
  holds ComplRelStr G = G;

theorem :: NECKLA_3:16
  for G being strict irreflexive RelStr holds ComplRelStr (ComplRelStr G) = G;
```

```
theorem :: NECKLA_3:17
  for G1,G2 being RelStr st the carrier of G1 misses the carrier of G2
  holds ComplRelStr union_of(G1,G2) = sum_of(ComplRelStr G1, ComplRelStr G2);

theorem :: NECKLA_3:18
  for G1,G2 being RelStr st the carrier of G1 misses the carrier of G2
  holds ComplRelStr sum_of(G1,G2) = union_of(ComplRelStr G1, ComplRelStr G2);

theorem :: NECKLA_3:19
  for G being RelStr, H being full SubRelStr of G
  holds the InternalRel of ComplRelStr H =
  (the InternalRel of ComplRelStr G)|_2 the carrier of ComplRelStr H;

theorem :: NECKLA_3:20
  for G being non empty irreflexive RelStr, x being Element of G,
  x' being Element of ComplRelStr G st x = x' holds
  ComplRelStr (subrelstr ([#]G \ {x})) = subrelstr ([#](ComplRelStr G) \ {x'});

begin :: Another facts relating to operation 'embeds'

registration
  cluster trivial strict -> N-free (non empty RelStr);
end;

theorem :: NECKLA_3:21
  for R being reflexive antisymmetric RelStr, S being RelStr holds
  (ex f being Function of R,S st for x,y being Element of R holds
  [x,y] in the InternalRel of R iff [f.x,f.y] in the InternalRel of S)
  iff S embeds R;

theorem :: NECKLA_3:22
  for G being non empty RelStr, H being non empty full SubRelStr of G
  holds G embeds H;

theorem :: NECKLA_3:23
  for G being non empty RelStr, H being non empty full SubRelStr of G
  st G is N-free holds H is N-free;

theorem :: NECKLA_3:24
  for G being non empty irreflexive RelStr holds
  G embeds Necklace 4 iff ComplRelStr G embeds Necklace 4;

theorem :: NECKLA_3:25
  for G being non empty irreflexive RelStr holds
  G is N-free iff ComplRelStr G is N-free;

begin :: Connected Graphs

definition
  let R be RelStr;
  mode path of R is RedSequence of the InternalRel of R;
end;

definition
  let R be RelStr;
```

```
  attr R is path-connected means
:: NECKLA_3:def 1

  for x,y being set st x in the carrier of R &
  y in the carrier of R & x <> y holds
  the InternalRel of R reduces x,y or the InternalRel of R reduces y,x;
end;

registration
  cluster empty -> path-connected RelStr;
end;

registration
  cluster connected -> path-connected (non empty RelStr);
end;

theorem :: NECKLA_3:26
  for R being non empty transitive reflexive RelStr, x,y being Element of R
  holds the InternalRel of R reduces x,y implies [x,y] in the InternalRel of R;

registration
  cluster path-connected -> connected (non empty transitive reflexive RelStr);
end;

theorem :: NECKLA_3:27
  for R be symmetric RelStr,x,y being set
  st x in the carrier of R & y in the carrier of R holds
  the InternalRel of R reduces x,y implies the InternalRel of R reduces y,x;

definition
  let R be symmetric RelStr;
  redefine attr R is path-connected means
:: NECKLA_3:def 2

  for x,y being set st x in the carrier of R &
  y in the carrier of R & x <> y holds (the InternalRel of R) reduces x,y;
end;

definition
  let R be RelStr;
  let x be Element of R;
  func component x -> Subset of R equals
:: NECKLA_3:def 3

  Class(EqCl the InternalRel of R, x);
end;

registration
  let R be non empty RelStr;
  let x be Element of R;
  cluster component x -> non empty;
end;

canceled;
```

```
theorem :: NECKLA_3:29
  for R being RelStr, x being Element of R,
  y be set st y in component x holds [x,y] in EqCl the InternalRel of R;

theorem :: NECKLA_3:30
  for R being RelStr, x being Element of R, A being set
  holds A = component x iff for y being set
  holds y in A iff [x,y] in EqCl the InternalRel of R;

theorem :: NECKLA_3:31
  for R be non empty irreflexive symmetric RelStr holds
  R is not path-connected implies
  ex G1,G2 being non empty strict irreflexive symmetric RelStr
  st the carrier of G1 misses the carrier of G2 &
  the RelStr of R = union_of(G1,G2);

theorem :: NECKLA_3:32
  for R be non empty irreflexive symmetric RelStr holds
  ComplRelStr R is not path-connected implies
  ex G1,G2 being non empty strict irreflexive symmetric RelStr
  st the carrier of G1 misses the carrier of G2 &
  the RelStr of R = sum_of(G1,G2);

theorem :: NECKLA_3:33
  for G being irreflexive RelStr st G in fin_RelStr_sp
  holds ComplRelStr G in fin_RelStr_sp;

theorem :: NECKLA_3:34
  for R be irreflexive symmetric RelStr
  st Card (the carrier of R) = 2 & the carrier of R in FinSETS
  holds the RelStr of R in fin_RelStr_sp;

theorem :: NECKLA_3:35
  for R be RelStr st R in fin_RelStr_sp holds R is symmetric;

theorem :: NECKLA_3:36
  for G being RelStr, H1,H2 being non empty RelStr,
  x being Element of H1, y being Element of H2
  st G = union_of(H1,H2) & the carrier of H1 misses the carrier of H2
  holds not [x,y] in the InternalRel of G;

theorem :: NECKLA_3:37
  for G being RelStr, H1,H2 being non empty RelStr,
  x being Element of H1, y being Element of H2
  st G = sum_of(H1,H2) holds not [x,y] in the InternalRel of ComplRelStr G;

theorem :: NECKLA_3:38
  for G being non empty symmetric RelStr, x being Element of G,
  R1,R2 being non empty RelStr st the carrier of R1 misses the carrier of R2
  & subrelstr ([#]G \ {x}) = union_of(R1,R2) & G is path-connected
  holds ex b being Element of R1 st [b,x] in the InternalRel of G;

theorem :: NECKLA_3:39
  for G being non empty symmetric irreflexive RelStr,
  a,b,c,d being Element of G, Z being Subset of G st Z = {a,b,c,d}
```

```
  & a,b,c,d are_mutually_different & [a,b] in the InternalRel of G
  & [b,c] in the InternalRel of G & [c,d] in the InternalRel of G
  & not [a,c] in the InternalRel of G & not [a,d] in the InternalRel of G
  & not [b,d] in the InternalRel of G holds subrelstr Z embeds Necklace 4;

theorem :: NECKLA_3:40
  for G being non empty irreflexive symmetric RelStr, x being Element of G,
  R1,R2 being non empty RelStr st the carrier of R1 misses the carrier of R2
  & subrelstr ([#]G \ {x}) = union_of(R1,R2) & G is non trivial
  & G is path-connected & ComplRelStr G is path-connected
  holds G embeds Necklace 4;

theorem :: NECKLA_3:41
  for G being non empty strict finite irreflexive symmetric RelStr
  st G is N-free & the carrier of G in FinSETS
  holds the RelStr of G in fin_RelStr_sp;
```

Listing C.3: The Mizar article `NECKLACE_3.abs`, the third article from the series, formalising "series-parallel" graphs of the original article [Tho00].

C.2 Formalisation of some properties of binary relations

```
:: Properties of First and Second Order Cutting of Binary Relations
::  by Krzysztof Retel
::
:: Received April 25, 2005
:: Copyright (c) 2005 Association of Mizar Users

environ

 vocabularies RELSET_2, TARSKI, RELAT_1, CANTOR_1, SETFAM_1, BOOLE, FUNCT_1,
      PUA2MSS1, EQREL_1, FUNCT_5, SUBSET_1, COMPLEX1;
 notations TARSKI, XBOOLE_0, ZFMISC_1, SUBSET_1, SETFAM_1, RELAT_1, FUNCT_1,
      RELSET_1, FUNCT_2, EQREL_1;
 constructors SETFAM_1, FUNCT_2, EQREL_1;
 registrations XBOOLE_0, SUBSET_1, RELAT_1, PARTFUN1;
 requirements SUBSET, BOOLE;


begin :: Preliminaries

:: Formalisation of first paragraph from the article:
:: "Relations binaries, fermetures, correspondances de Galois" (1948),
:: Prof. Jacques Riguet,
:: Bulettin de la S.M.F., tome 76 (1948), p.114-155.

reserve x,y,X,Y,A,B,C,M for set;
reserve P,Q,R,R1,R2 for Relation;

notation
  let X be set;
  synonym {_{X}_} for SmallestPartition X;
end;

theorem :: RELSET_2:1
  y in {_{X}_} iff ex x st y = {x} & x in X;

theorem :: RELSET_2:2
  X = {} iff {_{X}_} = {};

theorem :: RELSET_2:3
  {_{X\/Y}_} = {_{X}_} \/ {_{Y}_};

theorem :: RELSET_2:4
  {_{X/\Y}_} = {_{X}_} /\ {_{Y}_};

theorem :: RELSET_2:5
  {_{X\Y}_} = {_{X}_} \ {_{Y}_};

theorem :: RELSET_2:6
  X c= Y iff {_{X}_} c= {_{Y}_};
```

```
theorem :: RELSET_2:7
  for B1, B2 being Subset-Family of M holds
  Intersect(B1) /\ Intersect(B2) c= Intersect(B1 /\ B2);

theorem :: RELSET_2:8 :: (27.2)
  (P /\ Q)*R c= (P*R) /\ (Q*R);

begin :: The first order cutting of binary relation of two sets A,B

:: under a subset of the set A

definition
  canceled;
end;

theorem :: RELSET_2:9
  y in Im(R,x) iff [x,y] in R;

theorem :: RELSET_2:10
  Im(R1 \/ R2,x) = Im(R1,x) \/ Im(R2,x);

theorem :: RELSET_2:11
  Im(R1 /\ R2,x) = Im(R1,x) /\ Im(R2,x);

theorem :: RELSET_2:12
  Im(R1 \ R2,x) = Im(R1,x) \ Im(R2,x);

theorem :: RELSET_2:13
  (R1 /\ R2).:{_{X}_} c= R1.:{_{X}_} /\ R2.:{_{X}_};

definition
  let X,Y be set;
  let R be Relation of X,Y;
  let x be set;
  redefine func Im(R,x) -> Subset of Y;
  redefine func Coim(R,x) -> Subset of X;
end;

theorem :: RELSET_2:14
  for A being set, F being Subset-Family of A, R be Relation holds
  R.: union F = union {R.:X where X is Subset of A: X in F};

:: (3.1.2) - RELAT_1:149

theorem :: RELSET_2:15
  for A being non empty set, X being Subset of A holds
  X = union {{x} where x is Element of A: x in X};

theorem :: RELSET_2:16
  for A being non empty set, X being Subset of A holds
  {{x} where x is Element of A: x in X} is Subset-Family of A;

theorem :: RELSET_2:17 :: R(X) - original counterpart. The First Order Cutting.
  for A being non empty set, B being set,
  X being Subset of A, R being Relation of A,B
```

```
  holds R.:X = union {Class(R,x) where x is Element of A: x in X};

theorem :: RELSET_2:18
  for A being non empty set, B being set, X being Subset of A,
  R being Relation of A,B holds
  {R.:x where x is Element of A: x in X} is Subset-Family of B;

definition
 canceled;
  let A be set, R be Relation;
  func .:(R,A) -> Function means
:: RELSET_2:def 3

  dom it = bool A & for X being set st X c= A holds it.X = R.:X;
end;

notation
  let B,A be set;
  let R be Subset of [:A,B:];
  synonym .:R for .:(R,A);
end;

theorem :: RELSET_2:19
  for A,B being set, R being Subset of [:A,B:] st
  X in dom(.:R) holds (.:R).X = R.:X;

theorem :: RELSET_2:20
  for A,B being set, R being Subset of [:A,B:] holds rng(.:R) c= bool rng R;

theorem :: RELSET_2:21
  for A,B being set, R being Subset of [:A,B:]
  holds .:R is Function of bool A, bool rng R;

definition
  let B,A be set;
  let R be Subset of [:A,B:];
  redefine func .:R -> Function of bool A, bool B;
end;

theorem :: RELSET_2:22 :: FUNCT_3:15
  for A,B being set, R being Subset of [:A,B:] holds
  union((.:R).:A) c= R.:(union A);

begin :: The second order cutting of binary relation of two sets A,B

:: under a subset of the set A

reserve X,X1,X2 for Subset of A;
reserve Y for Subset of B;
reserve R,R1,R2 for Subset of [:A,B:];
reserve F for Subset-Family of A;
reserve FR for Subset-Family of [:A,B:];

definition
  let A,B be set, X be Subset of A, R be Subset of [:A,B:];
```

```
  func R.:^X equals
:: RELSET_2:def 4

  Intersect(.:R.:{_{X}_});
end;

definition
  let A,B be set;
  let X be Subset of A;
  let R be Subset of [:A,B:];
  redefine func R.:^X -> Subset of B;
end;

theorem :: RELSET_2:23
  .:R.:{_{X}_} = {} iff X = {};

theorem :: RELSET_2:24
  y in R.:^X implies for x being set st x in X holds y in Im(R,x);

theorem :: RELSET_2:25
  for B being non empty set, A being set, X being Subset of A,
  y being Element of B, R being Subset of [:A,B:] holds
  y in R.:^X iff for x being set st x in X holds y in Im(R,x);

theorem :: RELSET_2:26
  (.:R).:{_{X1}_} = {} implies R.:^(X1\/X2) = R.:^X2;

:: ksiazka S o R = SR a w MML jest to R*S
:: (1) S .:(R.:X) = (S o R).:X
:: w notacji uzytej w MML:  S.:(R.:X) = (R*S).:X

theorem :: RELSET_2:27 :: (2)
  R.:^(X1\/X2) = (R.:^X1) /\ (R.:^X2);

theorem :: RELSET_2:28
  for A being non empty set, B being set, F being Subset-Family of A,
  R being Relation of A,B
  holds {R.:^X where X is Subset of A: X in F} is Subset-Family of B;

theorem :: RELSET_2:29  :: (3.2.2)
  X = {} implies R.:^X = B;

canceled;

theorem :: RELSET_2:31 :: (3.2.1)
  for A being set,B being non empty set, R being Relation of A,B,
  F being Subset-Family of A, G being Subset-Family of B st
  G = {R.:^Y where Y is Subset of A: Y in F} holds R.:^(union F) = Intersect G;

theorem :: RELSET_2:32  :: (4)
  X1 c= X2 implies R.:^X2 c= R.:^X1;

theorem :: RELSET_2:33 :: (5)
  R.:^X1 \/ R.:^X2 c= R.:^(X1/\X2);
```

```
theorem :: RELSET_2:34 :: (6)
  (R1 /\ R2).:^X = (R1.:^X) /\ (R2.:^X);

theorem :: RELSET_2:35 :: (7.1.1)
  (union FR).:X = union {R.:X where R is Subset of [:A,B:]: R in FR};

:: (7.1.2) - RELAT_1:150

theorem :: RELSET_2:36
  for FR being Subset-Family of [:A,B:], A,B being set,
  X being Subset of A holds
  {R.:^X where R is Subset of [:A,B:]: R in FR} is Subset-Family of B;

:: (11.2) theorem TH40.

theorem :: RELSET_2:37  :: (11.1)
  R = {} & X <> {} implies R.:^X = {};

theorem :: RELSET_2:38  :: (7.2.2)
  R = [:A,B:] implies R.:^X = B;

theorem :: RELSET_2:39 :: (7.2.1)
  for G being Subset-Family of B st
  G = {R.:^X where R is Subset of [:A,B:]: R in FR} holds
  (Intersect FR).:^X = Intersect G;

theorem :: RELSET_2:40 :: (8)
  R1 c= R2 implies R1.:^X c= R2.:^X;

theorem :: RELSET_2:41 :: (9)
  (R1.:^X) \/ (R2.:^X) c= (R1 \/ R2).:^X;

theorem :: RELSET_2:42
  y in Im(R‘,x) iff not [x,y] in R & x in A & y in B;

theorem :: RELSET_2:43 :: (17)
  X <> {} implies R.:^X c= R.:X;

theorem :: RELSET_2:44
  for X, Y being set holds
  X meets R~.:Y iff ex x,y being set st x in X & y in Y & x in Im(R~,y);

theorem :: RELSET_2:45
  for X, Y being set holds
  (ex x,y being set st x in X & y in Y & x in Im(R~,y)) iff Y meets R.:X;

theorem :: RELSET_2:46  :: (19)
  X misses R~.:Y iff Y misses R.:X;

theorem :: RELSET_2:47  :: (14.1)
  for X being set holds R.:X = R.:(X /\ proj1 R);

theorem :: RELSET_2:48  :: (14.2)
  for Y being set holds (R~).:Y = (R~).:(Y /\ proj2 R);
```

```
theorem :: RELSET_2:49  :: (10.2)
  (R.:^X)` = R`.:X;

reserve R for Relation of A,B;
reserve S for Relation of B,C;

definition
  let A,B,C be set;
  let R be Subset of [:A,B:], S be Subset of [:B,C:];
  redefine func R*S -> Relation of A,C;
end;

theorem :: RELSET_2:50  :: (10.1)
  (R.:X)` = (R`).:^X;

:: (12) - FUNCT_5:20, RELAT_1:37

theorem :: RELSET_2:51
  proj1 R = (R~).:B & proj2 R = R.:A;

theorem :: RELSET_2:52 :: (13.1)
  proj1 (R*S) = (R~).:(proj1 S) & proj1 (R*S) c= proj1 R;

theorem :: RELSET_2:53 :: (13.2)
  proj2 (R*S) = S.:(proj2 R) & proj2 (R*S) c= proj2 S;

theorem :: RELSET_2:54 :: (15.1)
  X c= proj1 R iff X c= (R*(R~)).:X;

theorem :: RELSET_2:55 :: (15.2)
  Y c= proj2 R iff Y c= ((R~)*R).:Y;

theorem :: RELSET_2:56
  proj1 R = (R~).:B & (R~).:(R.:A) = (R~).:(proj2 R);

theorem :: RELSET_2:57 :: (16.1)
  (R~).:B = (R*(R~)).:A;

theorem :: RELSET_2:58 :: (16.2)
  R.:A = (R~*R).:B;

theorem :: RELSET_2:59 :: (18)
  S.:^(R.:X) = (R*S`)`.:^X;

theorem :: RELSET_2:60  :: (24.3)
  (R`)~ = (R~)`;

theorem :: RELSET_2:61  :: (20)
  X c= (R~).:^Y iff Y c= R.:^X;

theorem :: RELSET_2:62 :: (21)
  R.:(X`) c= Y` iff R~.:Y c= X;

theorem :: RELSET_2:63 :: (22)
  X c= (R~).:^(R.:^X) & Y c= R.:^((R~).:^Y);
```

```
theorem :: RELSET_2:64 :: (23)
  R.:^X = R.:^( R~.:^(R.:^X) ) & R~.:^Y = (R~).:^(R.:^((R~).:^Y));

theorem :: RELSET_2:65 :: (29.3)
  (id A)*R = R*(id B);

:: (24.1) - RELAT_1:40
:: (24.2) - RELAT_1:39
:: (24.4) - RELAT_1:54
:: (24.5) - R1 c= R2 implies R1~ c= R2~    :: SYSREL:27
:: (24.6) - R~~ = R; atomatically
:: (25.1) - RELAT_1:51
:: (25.2) - (S1 \/ S2)*R = S1*R \/ S2*R    :: SYSREL:20
:: (26) - RELAT_1:50
:: (27.1) - RELAT_1:52
:: (28.1) and (28.2) - RELAT_1:62
:: (29.1) and (29.2) - FUNCT_2:23
```

Listing C.4: The Mizar article `RELSET_2.abs`, the article formalising properties and collocation of "binary relations". The formalisation is based on the first chapter of the original article [Rig48] written by J. Riguet in 1948.

References

[ABFL05] S. Autexier, C. Benzmüller, A. Fiedler, and H. Lesourd. Integrating proof assistants as reasoning and verification tools into a scientific WYSIWIG editor. In *User Interfaces for Theorem Provers (UITP '05) [Workshop]*, Edinburgh, 2005.

[ABFL06] S. Autexier, C. Benzmüller, A. Fiedler, and H. Lesourd. Integrating proof assistants as plugins in a scientific editor. In *An Open Markup Format for Mathematical Documents, OMDoc (Version 1.2)* [Koh06b], pages 309–312.

[AFNW07] S. Autexier, A. Fiedler, T. Neumann, and M. Wagner. Supporting user-defined notations when integrating scientific text-editors with proof assistance. In MKM '07 [MKM07], pages 176–190.

[AR03] P. Audebaud and L. Rideau. TeXmacs as authoring tool for publication and dissemination of formal developments. In *Proceedings of the User Interfaces for Theorem Provers Workshop, UITP 2003*, volume 103 of *ENTCS*, pages 27–48, Rome, 2003.

[Aut] Automath archive. `http://automath.webhop.net/` Brouwer Institute in Nijmegen and the Formal Methods section of Eindhoven University of Technology.

[Ban90a] G. Bancerek. Cardinal numbers. *Formalized Mathematics*, 1(2):377–382, 1990.

[Ban90b] G. Bancerek. Tarski-grothendieck set theory as a basis of knowledge management system for mathematics, 1990. 36th Conference of History of Logic.

[Ban00] G. Bancerek. Development of the theory of continuous lattices in mizar. In Manfred Kerber and Mihael Kohlhase, editors, *Proceedings*

of the 8th Symposium on the Integration of Symbolic Computation and Mechanized Reasoning, pages 65–80. AK Peters Ltd., 2000.

[Ban03] G. Bancerek. On the structure of mizar types. *ENTCS*, 85(7):1–17, 2003.

[Bar06] H. Barendregt. *Informal*, page 10. Volume 3600 of Wiedijk [Wie06], 2006.

[BC04] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. An EATCS Series. Springer Verlag, May 2004.

[BCC+04] S. Buswell, O. Caprotti, D. P. Carlisle, M. C. Dewar, M. Gaëtano, and M. Kohlhase. *The OpenMath Standard*. The OpenMath Society, `http://www.openmath.org`, June 2004. Version 2.0.

[BM01] P. V. Biron and A. Malhotra. Xml schema part 2: Datatypes. Technical report, W3C, 2001. W3C Recommendation.

[BR02] G. Bancerek and P. Rudnicki. A compendium of continuous lattices in Mizar. *J. Automated Reasoning*, 29(3–4):189–224, 2002.

[BR03] G. Bancerek and P. Rudnicki. Information Retrieval in MML. In MKM '03 [MKM03], pages 119–132.

[BZ07] E. Borak and A. Zalewska. Mizar course in logic and set theory. In MKM '07 [MKM07], pages 191–204.

[CC99] O. Caprotti and A. M. Cohen. A type system for OpenMath. Technical report, The OpenMath Consortium, February 1999. RIACA, The Netherlands D1.3.2b.

[Com89] Library Committee. Mizar built-in notions. *Journal of Formalized Mathematics*, Axiomatics, 1989. `http://mizar.org/JFM/Axiomatics/hidden.html`.

[Com02] Library Committee. Boolean properties of sets — definitions, April 2002.

[Con04] W3C (World Wide Web Consortium). RDF Primer. W3C Recommendation, February 2004.

[Dav99] J. H. Davenport. A small OpenMath type system. Technical report, The OpenMath Consortium, April 1999. Bath D1.3.2c.

[Dav00] J. H. Davenport. A small OpenMath type system. *SIGSAM Bulletin (ACM Special Interest Group on Symbolic and Algebraic Manipulation)*, 34(2):16–21, 2000.

[Dav02] J. H. Davenport. On writing openmath content dictionaries. Technical report, The OpenMath Consortium, 2002. ESPIRIT project 24969.

[dB80] N.G. de Bruijn. A survey of the project Automath. In J.R. Hindley and J.P. Seldin, editors, *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalisms*, pages 579–606, Orlando, 1980. Academic Press. Reprinted in [NGdV94, A.5].

[dB87] N.G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In *Workshop on Programming Logic*, 1987. Reprinted in [NGdV94, F.3].

[dB91] N. G. de Bruijn. Checking mathematics with computer assistance. *Notices of the American Mathematical Society*, 38(1):8–15, 1991. Available at [Aut].

[FKF98] M. Flatt, S. Krishnamurthi, and M. Felleisen. Classes and mixins. In *Conf. Rec. POPL '98: 25th ACM Symp. Princ. of Prog. Langs.*, 1998.

[Gel04] G. Geleijnse. Comparing two user-friendly formal languages for mathematics: Weak type theory and mizar. Master's thesis, Technische Universiteit Eindhoven, May 2004.

[GHK+80] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. W. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices*. Springer-Verlag, 1980.

[Gru] T. Gruber. What is an ontology? `http://www-ksl.stanford.edu/kst/what-is-an-ontology.html`. last visit 2008-07-23.

[GS07] A. Grabowski and C. Schwarzweller. Revisions as an Essential Tool to Maintaing Mathematical Repositories. In MKM '07 [MKM07], pages 235–250.

[GW03] M. Giero and F. Wiedijk. MMode, a Mizar mode for the proof assistant Coq. Technical Report NIII-R0333, University of Nijmegen, 2003. Available at `http://www.cs.ru.nl/~freek/mmode/mmode.pdf` (last visited 2007–04–24).

[Har96] J. Harrison. A Mizar Mode for HOL. In Joakim von Wright, Jim Grundy, and John Harrison, editors, *Theorem Proving in Higher Order Logics: 9th International Conference, TPHOLs'96*, volume 1125 of *Lecture Notes in Computer Science*, pages 203–220, Turku, Finland, August 1996. Springer Verlag.

[HW80] G. H. Hardy and E. M. Wright. *An introduction to the Theory of Numbers*. Oxford University Press, 5th edition, April 1980.

[Jaś34] S. Jaśkowski. On the rules of supposition in formal logic. *Studia Logica*, 1934.

[JN04] G. Jojgov and R. Nederpelt. A path to faithful formalizations of mathematics. In MKM '04 [MKM04], pages 145–159.

[KB95] M. Kaufmann and R. S. Boyer. The boyer-moore theorem prover and its interactive enhancement. *Computers and Mathematics with Applications*, 29(2):27–62, 1995.

[KLMW07] F. Kamareddine, R. Lamar, M. Maarek, and J. B. Wells. Restoring natural language as a computerised mathematics input method. In MKM '07 [MKM07], pages 280–295.

[KMRW07a] F. Kamareddine, M. Maarek, K. Retel, and J. B. Wells. Gradual computerisation/formalisation of mathematical texts into Mizar. In Roman Matuszewski and Anna Zalewska, editors, *From Insight to Proof: Festschrift in Honour of Andrzej Trybulec*, volume 10(23) of *Studies in Logic, Grammar and Rhetoric*, pages 95–120. University of Białystok, 2007. Under the auspices of the Polish Association for Logic and Philosophy of Science.

[KMRW07b] F. Kamareddine, M. Maarek, K. Retel, and J. B. Wells. Narrative structure of mathematical texts. In MKM '07 [MKM07], pages 296–311.

[KMRW07c] F. Kamareddine, Manuel Maarek, Krzysztof Retel, and J. B. Wells. Digitised mathematics: Computerisation vs. formalisation. In *Review of the National Center for Digitization*, volume 10, pages 1–8, Faculty of Mathematics, Belgrade, Serbia, 2007.

[KMW04a] F. Kamareddine, Manuel Maarek, and J. B. Wells. Flexible encoding of mathematics on the computer. In MKM '04 [MKM04], pages 160–174.

[KMW04b] F. 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.

[KMW06] F. Kamareddine, Manuel Maarek, and J. B. Wells. Toward an object-oriented structure for mathematical text. In MKM '05 [MKM06], pages 217–233.

[KN04] F. Kamareddine and R. Nederpelt. A refinement of de Bruijn's formal language of mathematics. *J. Logic Lang. Inform.*, 13(3):287–340, 2004.

[Koh06a] M. Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.2).*, volume 4180 of *LNAI*. Springer Verlag, 2006.

[Koh06b] M. Kohlhase. *An Open Markup Format for Mathematical Documents, OMDoc (Version 1.2)*, volume 4180 of *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2006.

[KS76] M. Kordos and L. Szczerba. *Geometria dla nauczycieli (Geometry for Teachers)*. PWN, Warsaw, 1976.

[KW00] F. Kamareddine and J. B. Wells. Formath. A research proposal to UK funding body, 2000.

[KW01a] F. Kamareddine and J. B. Wells. MathLang: A new language for mathematics, logic, and proof checking. A research proposal to UK funding body, 2001.

[KW01b] F. Kamareddine and J. B. Wells. Promath presenting, proving, and programming mathemati cal books (case for support). 9 pages, acknowledgement G583082, December 2001.

[KW02] F. Kamareddine and J. B. Wells. MathLang: A new language for mathematics, logic, and proof checking (case for support). 9 pages, acknowledgement 2002102111273723614163, September 2002.

[KW08] F. Kamareddine and J. B. Wells. Computerizing mathematical text with MathLang. In Mauricio Ayala-Rincon and Heusler, editors, *Proc. Second Workshop on Logical and Semantic Frameworks, with Applications*, pages 5–30, Ouro Preto, Minas Gerais, Brazil, 2008. Elsevier. The LSFA '07 (post-event) proceedings is published as vol. 205 (2008-04-06) of *Elec. Notes in Theoret. Comp. Sci.*

[KWZ08] F. Kamareddine, J. B. Wells, and C. Zengler. Computerising mathematical text in mathlang. In Preparation, May 2008.

[Lan30] E. Landau. *Grundlagen der Analysis*. Chelsea, 1930.

[Lan51] E. Landau. *Foundations of Analysis*. Chelsea, 1951. Translation of [Lan30] by F. Steinhardt.

[Lan06] Ch. Lange. Swim – a semantic wiki for mathematical. Technical report, Jacob University Bremen, December 2006. Technical Report.

[Log06] LogiCal Project, INRIA, Rocquencourt, France. *The Coq Proof Assistant Reference Manual – Version 8.1*, 2006. Available at `ftp://ftp.inria.fr/INRIA/coq/V8.1/doc/`.

[LS99] O. Lassila and R. R. Swick. Resource description framework (rdf) model and syntax specification. Technical report, W3C, 1999. W3C recommendation.

[Maa07] M. Maarek. *Mathematical Documents Faithfully Computerised: the Grammatical and Text & Symbol Aspects of the MathLang Framework*. PhD thesis, Heriot-Watt University, Edinburgh, Scotland, June 2007.

[MG06] L. E. Mamane and H. Geuvers. A document-oriented Coq plugin for TeXmacs. In *Mathematical User-Interfaces Workshop 2006 [Workshop]*, Workingham, 2006.

[Miz] *Mizar Manuals*. `http://mizar.org/project/bibliography.html`.

[MKM03] *Mathematical Knowledge Management, 2nd Int'l Conf., Proceedings*, volume 2594 of *Lecture Notes in Computer Science*. Springer Verlag, 2003.

[MKM04] *Mathematical Knowledge Management, 3rd Int'l Conf., Proceedings*, volume 3119 of *Lecture Notes in Computer Science*. Springer Verlag, 2004.

[MKM06] *Mathematical Knowledge Management, 4th Int'l Conf., Proceedings*, volume 3863 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2006.

[MKM07] *Towards Mechanized Mathematical Assistants (Calculemus 2007 and MKM 2007 Joint Proceedings)*, volume 4573 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 2007.

[MKSM04] Ch. Welty M. K. Smith and D. L. McGuinness. Owl web ontology language guide. Technical report, W3C, 2004. W3C Recommendation.

[Mol07] J. M. Moller. General topology. Authors' notes, 2007.

[MR05] R. Matuszewski and P. Rudnicki. Mizar: the first 30 years. In Grzegorz Bancerek, editor, *30 Years of Mizar*, volume 4 of *Mechanized Mathematics and its Applications*, pages 3–24, March 2005. `http://merak.pb.bialystok.pl/mkm2004/`.

[MvH04] D. L. McGuinness and F. van Harmelen. Owl web ontology language overview. Technical report, W3C, 2004. W3C Recommendation.

[NB02] A. Naumowicz and Cz. Byliński. Basic elements of computer algebra in MIZAR. volume 2 of *Mechanized Mathematics and Its Applications*, pages 9–16, 2002.

[NB04] A. Naumowicz and Cz. Byliński. Improving MIZAR Texts with Properties and Requirements. In MKM '04 [MKM04], pages 290–301.

[Ned02] R. Nederpelt. Weak type theory: A formal language for mathematics. Technical report, Eindhoven University of Technology, May 2002.

[NGdV94] R. 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.

[NK01] 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.

[NNS04] K. Nakagawa, A. Nomura, and M. Suzuki. Extraction of logical structure from articles in mathematics. In MKM '04 [MKM04].

[NPW02] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer-Verlag, 2002.

[NS06] K. Nakagawa and M. Suzuki. Mathematical knowledge browser with automatic hyperlink detection. In MKM '05 [MKM06].

[ORS92] S. Owre, J. M. Rushby, and N. Shankar. PVS: A prototype verification system. In *The 11th International Conference on Automated Deduction*, volume 607 of *Lecture Notes in Computer Science*, pages 748–752, Saratoga Springs, NY, USA, June 1992. Springer Verlag.

[Ret03a] K. Retel. The class of series – parallel graphs. Part I. *Formalized Mathematics*, 11(1):99–103, 2003. Submitted to the MML at November 18, 2002.

[Ret03b] K. Retel. The class of series – parallel graphs. Part II. *Formalized Mathematics*, 11(3):289–291, 2003. Submitted to the MML at May 29, 2003. This work has been partially supported by CALCULEMUS grant HPRN-CT-2000-00102.

[Ret04] K. Retel. The class of series – parallel graphs. Part III. *Formalized Mathematics*, 12(2):143–150, 2004. Submitted to the MML at February 3, 2004.

[Ret05a] K. Retel. First year PhD report. Technical report, Heriot-Watt University, September 2005.

[Ret05b] K. Retel. Properties of first and second order cutting of binary relations. *Formalized Mathematics*, 13(3):361–365, 2005. Submitted to the MML at April 25, 2005.

[Rig48] J. Riguet. Relations binaires, fermetures, correspondances de galois. *Bulletin de la Socit Mathmatique de France*, 76:114–155, 1948. On WWW: `http://www.numdam.org/item?id=BSFM_1948__76__114_0`.

[Roy59] B. Roy. Transitivité et connexité. *C. R. Acad. Sci. Paris*, 249:216–218, 1959.

[RT99] P. Rudnicki and A. Trybulec. On equivalents of well-foundedness. an experiment in mizar. *Journal of Automated Reasoning*, 23(3–4):197–234, 1999.

[Rud92] P. Rudnicki. An overview of the Mizar project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, 1992.

[RV02] A. Riazanov and A. Voronkov. The design and implementation of Vampire. *AI Communications*, 15(2–3):91–110, 2002.

[RZ05] K. Retel and A. Zalewska. Mizar as a tool for teaching mathematics. In Grzegorz Bancerek, editor, *30 Years of Mizar*, volume 4 of *Mechanized Mathematics and its Applications*, pages 3–24, Bialowieża, Poland, March 2005. `http://merak.pb.bialystok.pl/mkm2004/`.

[Sie64] W. Sierpiński. *Elementary Theory of Numbers*, volume 42 of *Monografie Matematyczne*. Państwowe Wydawnictwo Naukowe, Warsaw, Poland, 1964. Translated from Polish by A. Hulanicki.

[Tar39] A. Tarski. On well-ordered subsets of any set. *Fundamenta Mathematicae*, 32:176–183, 1939.

[TB90] W. A. Trybulec and G. Bancerek. Kuratowski – Zorn lemma. *Formalized Mathematics*, 1(2):387–393, 1990.

[Tho00] S. Thomasse. On better-quasi-ordering countable series-parallel orders. *Transactions of the American Mathematical Society*, 352(6):2491–2505, 2000.

[Try77] A. Trybulec. Informationslogische sprache mizar, 1977. Heft 33m, Ilmenau.

[Try78] A. Trybulec. The mizar-qc/6000 logic information language. *ALLC Bulletin*, 6(2), 1978.

[Try80] A. Trybulec. The mizar logic information language. *Studies in Logic*, 1, 1980. Bialystok.

[Try82] A. Trybulec. Jezyk informacyjno logiczny mizar-mse. Technical Report 465, 1982. ICS PAS Reports, Warsaw.

[Try89] A. Trybulec. Tarski Grothendieck set theory. *Journal of Formalized Mathematics*, Axiomatics, 1989. `http://mizar.org/JFM/Axiomatics/tarski.html`.

[Urb06] J. Urban. MizarMode - an integrated proof assistance tool for the Mizar way of formalizing mathematics. *Journal of Applied Logic*, 4(4):414–427, 2006. available online at http://ktiml.mff.cuni.cz/∼urban/mizmode.ps.

[vBJ77] L. S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH System*. PhD thesis, Eindhoven, 1977. Partially reprinted in [NGdV94, B.5,D.2,D.3,D.5,E.2].

[vD80] D.T. van Daalen. *The Language Theory of Automath*. PhD thesis, Eindhoven University of Technology, 1980. Partially reprinted in [NGdV94, A.6,B.6,C.5].

[vdH01] J. van der Hoeven. GNU TeXmacs: A free, structured, WYSIWYG and technical text editor. *Cahiers GUTenberg*, 39–40:39–50, May 2001.

[vdH04] J. van der Hoeven. GNU TeXmacs. *SIGSAM Bulletin*, 38(1):24–25, 2004.

[Vor95] A. Voronkov. The anatomy of vampire: Implementing bottom-up procedures with code trees. *Journal of Automated Reasoning*, 15(2):237–265, 1995.

[WAB06] M. Wagner, S. Autexier, and C. Benzmüller. PLATO: A mediator between text-editors and proof assistance systems. In Christoph Benzmüller Serge Autexier, editor, *7th Workshop on User Interfaces for Theorem Provers (UITP'06)*, ENTCS. Elsevier, August 2006.

[War62] S. Warshall. A theorem on boolean matrices. *J. ACM*, 9(1):11–12, 1962.

[Wiea] F. Wiedijk. A proposed syntax for binders in mizar. `http://www.cs.ru.nl/~freek/notes/`.

[Wieb] F. Wiedijk. Writing a mizar article in nine easy steps. website `http://www.cs.ru.nl/~freek/notes/index.html`.

[Wie03] F. Wiedijk. Comparing mathematical provers. In MKM '03 [MKM03], pages 188–202.

[Wie04a] F. Wiedijk. Formal proof sketches. In *Proceedings of TYPES'03*, volume 3085 of *LNCS*, pages 378–393. Springer-Verlag, December 2004.

[Wie04b] F. Wiedijk. Formal proof sketches. In *Types for Proofs and Programs: Third International Workshop, TYPES 2003, LNCS 3085*, pages 378–393. Springer, 2004.

[Wie06] F. Wiedijk, editor. *The Seventeen Provers of the World, foreword by Dana S. Scott*, volume 3600 of *LNCS*. Springer Berlin, Heidelberg, 2006.

[WM99] N. Walsh and L. Muellner. *DocBook: The Definitive Guide*. OŔeilly & Associates, Inc., 1st edition edition, October 1999.

Author Index

Index