MathLang: experience-driven development of a new mathematical language

Fairouz Kamareddine[1] Manuel Maarek[1] J. B. Wells[1]

ULTRA Group
Heriot-Watt University
Edinburgh, Scotland

Abstract

In this paper we report on the design of a new mathematical language and our method of designing it, driven by the encoding of mathematical texts. MathLang is intended to provide support for checking basic well-formedness of mathematical text without requiring the heavy and difficult-to-use machinery of full type theory or other forms of full formalization. At the same time, it is intended to allow the addition of fuller formalization to a document as time and effort permits. MathLang is intended to, ultimately, be useful in providing better software support for authoring mathematics, reading mathematics, and organizing and distributing mathematics. The preliminary language presented in this paper is intended only for machine manipulation and for debugging of the design of MathLang.

Key words: Mathematical language, Mathematical vernacular, Mathematical knowledge management Weak types, MathLang

1 Introduction

Data management has become an important area for automation. Editing, storage, publishing, data retrieving and other computations are gratefully helped by computers with appropriate software. Nowadays computers could be used at each step of writing texts, the use of pen & paper may not be essential. Could we make the same remark for mathematical work? Would it be possible for a mathematician to use computers as a help tool from scratch to treatise?

[1] http://www.macs.hw.ac.uk/ultra/

1.1 *Current situation of mathematics on computers*

Putting mathematics on computers.

First of all, to use computers for mathematical purposes, we need to put mathematical content inside them. Different ways to do so exist:

- One can *scan* mathematical books and store their images on electronic support. This solution brings no facilities of computation and even no automatic search on data.
- One can *encode* mathematics. Programs could then deal with these encoded mathematical data. Storage is possible, and so are computations on data such as visualizing, calculating, analysing and searching.

Existing encodings.

Many languages to encode mathematics exist already. We sort them in categories based on what they were invented for:

A. Languages for printing mathematical symbols on paper/screen (e.g. LaTeX). Since they follow a *rendering* aim, they encode only the shape of a document and not its meaning.

B. Languages for theorem provers and computer algebra systems (see [Wie03]) which try to formalize mathematics. These systems claim to assist the mathematician to prove theorems by verifying them.

C. Languages which store the semantical structure of mathematical texts without checking it (e.g. OMDoc [Koh03]). They combine natural language with notions like formulae or text structures (theorems, examples,...).

Use of encoded texts.

Choosing an encoding depends on its intended use. Hence we need to know what computers could do to assist the mathematician in his work. Mathematical work could be summarized in three levels:

1. At the first one, the mathematician starts from scratch to put down more or less random ideas. He organizes, polishes and refines them.
2. At the second level, ideas evolve into clearer views and theories begin to take shape. Things move to a concrete form that allows publications.
3. At the last level, calculations and proofs take place. Details need to be clarified to reach complete theories. This level leads to a full formalization.

At each of these levels we deal with mathematical data. From structured but not complete content in the first level to formalized data at the latter. None of the encoding listed above satisfies the needs of all these levels:

- Languages for rendering (A) do not capture the semantics of a text (for use in level 3).

- Languages in (B) are too strict to encode incomplete (from level 1) or partly-formalized (from level 3) mathematical contents.
- Languages in (C) do not allow enough automation to benefit from computers at each level.

Thus we gather that currently there is no language to encode mathematics to be used at every level of mathematical work.

1.2 Our Concerns

N.G. de Bruijn in his extensive writings and different stages of mathematical languages and vernaculars proposed that "the way mathematical material is to be presented to the [computer] system should correspond to the usual way we write mathematics." We are proposing a language, MathLang to realise this proposition. MathLang goals could be described following four main concerns for mathematics on computers.

- The first concern is to be able to automate computations of data encoded in MathLang to use computer skills. For this reason we have designed MathLang using a *full symbolism* (that is to say every element of the language consists of recognisable symbols). This facilitates the manipulations that can be done by the computer.
- When using existing formal languages to write mathematics, one needs to be sure that the mathematics used will fit in the underlying logic of the language. This restricts the expressivity of the language. To cover all mathematics, MathLang is intended to describe the *structure of mathematical texts* and their *reasoning steps*.
- Having a grammatical encoding for mathematical texts brings obviously the need to have properties of these texts. A *type system* validates the grammatical structure of a MathLang text. A sentence like "if x belongs to $\mathfrak{M}$ then $x + y = y + x$" will be valid if x, y, $\mathfrak{M}$, "belongs to", "if", $+$ and $=$ are known beforehand and satisfy some weak typing relations. Otherwise, "if x belongs to $\mathfrak{M}$ then $x + y$" does not make sense and is pointed as incorrect with the MathLang type system.
- Our last important concern is to provide a user interface for the working mathematician. Providing *one language* which allows to encode any mathematics and to leave the possibility of further transformations through more formal data, will make the step of bridging mathematics and computers as smooth as possible.

1.3 From MV to WTT to MathLang

In 1979, de Bruijn developed a course on the Mathematical Vernacular (MV) intended to be a language to write mathematics. This course became part of the curriculum for mathematics teachers in the Netherlands. In 1987, almost twenty years after the beginning of Automath, de Bruijn published an article on MV (cf. F3 of [Aut94]). In MV, a Mathematical text is seen as a set of lines. Each line being either the introduction of a new notion or an assumption, these in a certain context. The structure of a text is line-by-line where each notion used in a line should have been defined beforehand. This structure was inspired by the development of the language Automath. In MV a notion of typing gives meta-information in the language. Two levels of typing are described: the *low typing* which expresses that an object is part of a certain set and the *high typing* which indicates the grammatical category of which an expression is part. All this makes MV:

- faithful to the mathematician's language while being formal and avoiding ambiguities;
- close to the way in which mathematicians express themselves in writing;
- possess a syntax based on linguistic categories rather than sets/types;
- mainly concerned with structural rather than logical correctness.

The type theory of MV is *weak* because it is composed mostly by atomic types. These types refer to the grammatical categories of a Mathematical text. Using the rules of MV one can check if the reasoning structure of a text is valid. After de Bruijn's retirement, Nederpelt took over the course and continues to teach it today. Nederpelt refined MV into the so-called Weak Type Theory (WTT) which has a precise abstract syntax. This makes it possible to establish important desirable properties such as strong normalisation, decidability of type checking and subject reduction as was done by Kamareddine. For details on MV and WTT, see [Aut94,KN04].

Since MV and WTT are said to help provide a language to encode mathematics which can be used at every level of mathematical work, we set out to test these languages through the *Foundations of Analysis* (E. Landau [Lan51]) which is already fully formalised in Automath [vBJ77] by Bert van Benthem-Jutting. Since our aim is not the full formalisation, but an encoding which allows a full formalisation at a later stage, we felt that choosing a fully formalised book for our encoding, would allow us in the future, to compare the already existing full formalisation, with the full formalisation that could be built on top of our encoding. This paper reports the extensions that had to be made to WTT to encode the first chapter while remaining faithful to the mathematician's intentions and keeping the road open to reach (in the long run) a full formalisation that can be compared to that of Automath. In particular, we present the language MathLang, the implementation we have made of its type checker and an overview of the MathLang translation we have done of the first chapter of E. Landau [Lan51]. In Section 2, we give the abstract syntax of MathLang. To illustrate the use and extensions of MathLang we give examples taken from our translation of the first chapter of [Lan51]. In Section 3, we describe the derivation rules of MathLang. In Section 4 we discuss our implementation of MathLang and present the full translation of the first chapter which is automatically checked by our software. In Section 5 we discuss related and future work and we conclude.

2 Abstract syntax of MathLang

MathLang is an extension of both MV and WTT. It attempts to be closer to a grammatical encoding of the reasoning structure of Mathematics. MathLang is designed to encoded entire mathematical texts. It is composed by several *grammatical levels* to distinguish mathematical structures from symbols to entire books. The levels define *grammatical categories* as groups of mathematical objects. In this section we will first explain what we mean by grammatical categories (Section 2.1) and then describe the grammatical levels (Section 2.2). The abstract syntax is presented here with examples of MathLang encodings taken from our translation of the first chapter of E. Landau's *Foundations of Analysis* [Lan51].

2.1 Grammatical categories

MathLang extends the grammatical categories of WTT which in turn extends those of MV.

`T` We first have a grammatical category that groups the so-called *terms* which are common mathematical objects like "$x+1$", "the point A", or "a triangle ABC". `T` denotes the set of terms.

`S` Then we have the sets of mathematical objects like "$\mathbb{N}$" (the set of natural numbers). We name this category *sets* and use `S` for the set of *sets*.

`N` The *nouns* grammatical category is commonly used in mathematics to designate families of terms. For example in the sentence "1 is a natural number", "natural number" is a noun. The set of nouns is represented by `N`.

`A` In MathLang, a noun could be defined from another noun. For example, "isosceles triangle" is a noun. It is a restricted family of "triangle". We define *adjectives* as the kind of expressions that refine and/or change the meaning of a noun. "Isosceles" is then an element of the set `A` of *adjectives*.

`P` Expressions like "$x = 1$" or "$\forall x \in \mathbb{N}, x \geq 1$" which describe mathematical properties are *statements* in MathLang. `P` is the set of *statements*.

`D` The sentences that define new symbols in mathematical texts are called *definitions* in MathLang. For example, "We define $x + y$ such that ..." belongs to the set `D` of *definitions*.

`Z` The grammatical category of *declarations* groups variable declarations like "let $\mathfrak{M}$ be a set". Elements of the set of declaration `Z` could be components of contexts (see below).

Γ, Γ_F and Γ_{FS} Construction like "let x be in $\mathfrak{M}$", "assume that $y > x + 1$ ", etc., are elements that build a context. These are declarations and assumptions needed before stating properties. Combinations of this kind of expressions are named *contexts* and belong to the set Γ. Furthermore, for assumptions and declarations which cover a certain paragraph, we have the notion of flags (see Example 2.3).

`L` *Lines* in MathLang are steps of reasoning in mathematical texts. They sometimes correspond to real lines in texts.

`K` Proofs, examples, paragraphs or sections are groups of lines. We designate them by *blocks*. `K` is the set of blocks.

`B` The grammatical category of *books* designates MathLang documents. A mathematical text corresponds to a *book* and so is an element of `B`.

Figure 1 shows an example of a mathematical text with a diagram of the corresponding MathLang structure. MathLang can be seen as providing for a mathematical text, a grammatical structure that is not necessarily what would result from an analysis by linguists. Section 3 explains how one can check the well-formedness of a MathLang structure. Although the long-term goals of MathLang include the integration with mathematical texts written in natural language, we do not yet do this — the figure is purely to help understand the eventual role we expect MathLang to play. We do not yet have any mechanism for matching a MathLang structure with actual natural language mathematical text.

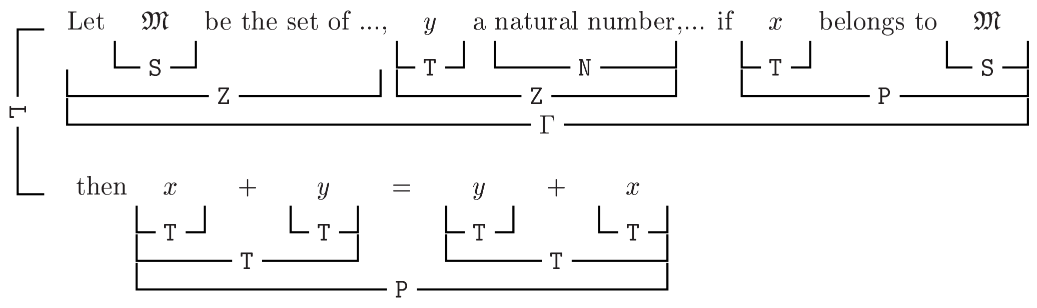
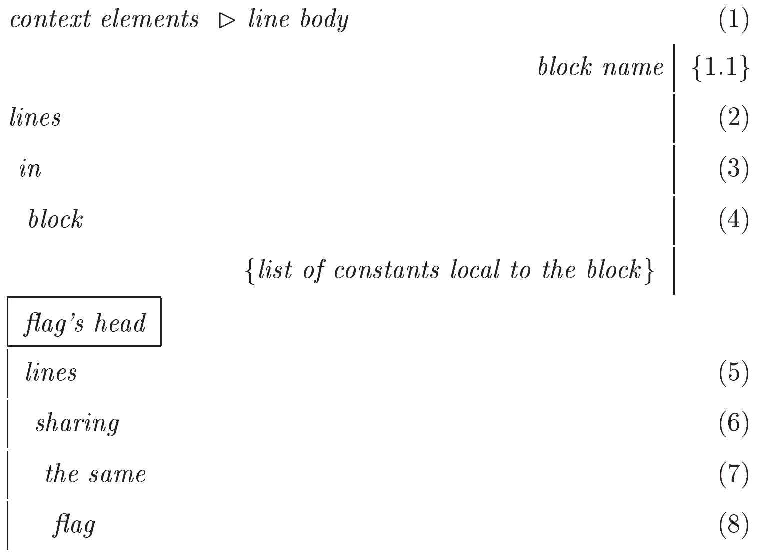

Let 𝔐 be the set of ..., y a natural number,... if x belongs to 𝔐
S
T
N
T
S
Z
Z
P
Γ
L
then x + y = y + x
T
T
T
T
T
T
P

Fig. 1. A mathematical line and its MathLang grammatical categories

2.2 *Grammatical levels*

In this section we describe the four grammatical levels of MathLang: *atomic*, *phrase*, *sentence* and *discourse*. We extend the elements of the third and fourth levels of WTT. We will illustrate these extensions with examples taken from our translation of the first chapter of [Lan51] (see the authors' web pages for the translation of the full chapter). Examples 2.2, 2.3, 2.4 and 2.5 illustrate MathLang constructions: flags, definitions by cases, blocks and references. As to the notations we use to print MathLang texts, note that the language MathLang is still in development since the translations of the other chapters will no doubt call for further extensions. For this reason the MathLang texts given here are rendered with an experimental syntax which we intend to improve. Currently, the concrete syntax of MathLang is implemented using the XML recommendations. We have made this choice to facilitate transformations on MathLang data. One of these transformations produces automatically the renderings shown in this paper. Below is an example. *(i)* being line numbers and $\{i.i\}$ being block indices. In a line, the symbol $\triangleright$ separates the context from the new statement or the new definition.



(i) The *atomic level* is composed by identifiers: the mathematical symbols. There are three kinds of identifiers: variables (undefined mathematical objects), constants (defined objects) and binders (to write constructions which locally introduce a variable).

$$\text{Variables } v \in V \qquad \text{Constants } c \in C \qquad \text{Binders } b \in B$$

Whereas constants and binders can be of any grammatical category, a variable can only be a term or a set or a statement. We use superscripts to denote the relevant category. For example, $v \in V^{\mathtt{T}}$ means that v is a variable of category $\mathtt{T}$.

Example 2.1 Identifiers

In an expression (taken from our translation of the first chapter of [Lan51]) like $\forall x : \mathfrak{M}, x + 1 = x'$:

- x, $\mathfrak{M}$ are variables
- $:$, 1, $+$, $=$ and $'$ are constants
- $\forall$ is a binder

(ii) The *phrase level* is the formula level. It describes how to construct terms, sets, nouns and adjectives. Four constructions exists. Variable instantiations V. Constant calls $C(\vec{\mathtt{E}})$. Abstractions with binders $B_{\mathtt{Z}}(\mathtt{E})$. And attributions $\mathtt{AN}$ that attribute an adjective to a noun to create a new noun.

$$\begin{array}{llcll}
\text{Terms} & \mathtt{t} & \in & \mathtt{T} & ::= \; V^{\mathtt{T}} \mid C^{\mathtt{T}}(\vec{\mathtt{E}}) \mid B^{\mathtt{T}}_{\mathtt{Z}}(\mathtt{E}) \\
\text{Sets} & \mathtt{s} & \in & \mathtt{S} & ::= \; V^{\mathtt{S}} \mid C^{\mathtt{S}}(\vec{\mathtt{E}}) \mid B^{\mathtt{S}}_{\mathtt{Z}}(\mathtt{E}) \\
\text{Nouns} & \mathtt{n} & \in & \mathtt{N} & ::= \; C^{\mathtt{N}}(\vec{\mathtt{E}}) \mid B^{\mathtt{N}}_{\mathtt{Z}}(\mathtt{E}) \mid \mathtt{AN} \\
\text{Adjectives} & \mathtt{a} & \in & \mathtt{A} & ::= \; C^{\mathtt{A}}(\vec{\mathtt{E}}) \mid B^{\mathtt{A}}_{\mathtt{Z}}(\mathtt{E})
\end{array}$$

We take E and Z as below. We also take $\vec{\mathtt{E}}$ to be a list of E's.

$$\begin{array}{llcll}
\text{Expressions} & \mathtt{e} & \in & \mathtt{E} & ::= \; \mathtt{T} \mid \mathtt{S} \mid \mathtt{N} \mid \mathtt{P} \\
\text{Declarations} & \mathtt{z} & \in & \mathtt{Z} & ::= \; V^{\mathtt{T}} : \mathtt{S} \mid V^{\mathtt{T}} : \mathtt{N} \mid V^{\mathtt{S}} : \mathrm{SET} \mid V^{\mathtt{P}} : \mathrm{STAT}
\end{array}$$

The set Z describes the four constructions which declare a new variable. In the first two constructions, the colon states the belonging of the declared variable. The variable on the left side of the colon belongs to the entity on the right side of the colon. The first construction introduces a term variable by stating in which set the term belongs (expression from S). The second also introduces a term variable but by stating of which noun it is an instance of (expression from N). With the keyword : SET (resp. : STAT), the third (resp. fourth) construction introduces a new set (resp. statement) variable.

(iii) The *sentence level* defines how to construct one step of reasoning: either statement or definition. Note the extension with definitions by cases in $C(\vec{\mathtt{E}}) := \mathtt{E}$.

$$\begin{array}{llcllll}
\text{Statements} & \mathtt{p} & \in & \mathtt{P} & ::= & V^{\mathtt{P}} \mid C^{\mathtt{P}}(\vec{\mathtt{E}}) \mid B^{\mathtt{P}}_{\mathtt{Z}}(\mathtt{E}) & \\
\text{Definitions} & \mathtt{d} & \in & \mathtt{D} & ::= & C^{\mathtt{T}}(\vec{V}) := \mathtt{T} & \mid C^{\mathtt{T}}(\vec{\mathtt{E}}) := \mathtt{T} \\
 & & & & \mid & C^{\mathtt{S}}(\vec{V}) := \mathtt{S} & \mid C^{\mathtt{S}}(\vec{\mathtt{E}}) := \mathtt{S} \\
 & & & & \mid & C^{\mathtt{N}}(\vec{V}) := \mathtt{N} & \mid C^{\mathtt{N}}(\vec{\mathtt{E}}) := \mathtt{N} \\
 & & & & \mid & C^{\mathtt{A}}(\vec{V}) := \mathtt{A} & \mid C^{\mathtt{A}}(\vec{\mathtt{E}}) := \mathtt{A} \\
 & & & & \mid & C^{\mathtt{P}}(\vec{V}) := \mathtt{P} & \mid C^{\mathtt{P}}(\vec{\mathtt{E}}) := \mathtt{P}
\end{array}$$

As we see, the set of definitions D is composed by two kinds of constructions. The first, $C(\vec{V}) := \mathtt{E}$ (representing constructions of the left column) gives an expression as a full definition for the constant. Each parameter of the constant is a variable (these parameters are represented as a list of variables: $\vec{V}$). The second, $C(\vec{\mathtt{E}}) := \mathtt{E}$ (representing constructions of the right column) is used to define a constant with several cases (by providing each time an expression). Each case of the definition will be described using one definition line (see the lines in the discourse level). The parameters are now pattern expressions that match the case ($\vec{\mathtt{E}}$). MathLang type analysis (see Section 3) checks that variables appearing in the patterns are well declared in the context. MathLang checking per-forms only this kind grammatical analysis. Note that no comparisons of cases to find uncovered cases or unused ones are done while this is a more semantical analysis.

Example 2.2 Definitions by cases
A common way to define mathematical objects is to use cases. As an example we take the following text from Landau's Definition 1 of Chapter 1, Section 2:

> **Theorem 4**, *and at the same time* **Definition 1:** *To every pair of numbers x, y, we may assign in exactly one way a natural number, called $x + y$* (+ to be read "plus"), *such that*
>
> $$x + 1 = x' \text{ for every } x \tag{1}$$
>
> $$x + y' = (x + y)' \text{ for every } x \text{ and every } y \tag{2}$$
>
> *$x + y$ is called the sum of x and y, or the number obtained by addition of y to x.* *[Lan51]*

We only consider the definition of the + operator which is recursively defined by two equations. The definition by cases we have introduced in MathLang gives an encoding of this kind of mathematical definitions. This encoding is closer to the original text than was the normal encoding in MV and WTT (that is to say, providing a unique object as definition). This normal encoding of MV and WTT can still be made in MathLang as follows:

$$x : \mathbb{N}, y : \mathbb{N} \quad \rhd +(x, y) := \iota_{z:\mathbb{N}}\, ((y = 1 \implies z = x') \wedge \exists_{t:\mathbb{N}}\, (y = t' \implies z = (x + t)'))$$

This MathLang text defines $x + y$ in one line. The two cases of the original text are represented by a conjunction of two implications. This encoding however is not close enough to the original text. The original definition of + is explicitly composed by two cases while here it is merged in one case using the logical symbols of implication $\implies$ and conjunction $\wedge$ which are not properly defined in E. Landau's text.

Let us see now how the same definition of the addition of natural numbers could be expressed in MathLang using, this time, a *definition by cases*. As it is written in E. Landau's text, in both cases we *assign* a term to the addition of a pair of terms:

$$\begin{array}{r|c} \textit{Definition 1} & \{2.4\} \\ x : \mathbb{N} \rhd +(x, 1) := x' & (38) \\ x : \mathbb{N},\ y : \mathbb{N} \rhd +(x, y') := (x + y)' & (39) \end{array}$$

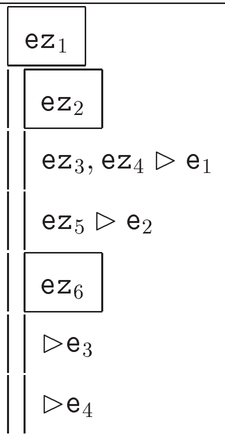
In MathLang, a definition by cases defines a constant using several lines. Each line being one case. This construction brings MathLang closer to the original text for two reasons. First, to each original case corresponds one case in MathLang. Second, in this encoding we are not using additional constants. We do not need logical symbols as we did in the earlier translation of MV and WTT. These logical constructions may not always be explicitly defined in the mathematical text.

(iv) The *discourse level* gives constructions to describe the structure of mathematical texts. Note the extension with flags, flagstaffs and blocks.

$$\begin{array}{lcccll}
\text{Contexts} & \gamma & \in & \Gamma & ::= & \Gamma_F \mid \Gamma, \mathtt{Z} \mid \Gamma, \mathtt{P} \\
\text{Flags} & \gamma_F & \in & \Gamma_F & ::= & \Gamma_{FS} \mid \Gamma_F, [\mathtt{Z}] \mid \Gamma_F, [\mathtt{P}] \\
\text{Flagstaffs} & \gamma_{FS} & \in & \Gamma_{FS} & ::= & \emptyset_\Gamma \mid \Gamma_{FS}, \bullet \\
\text{Lines} & \mathtt{l} & \in & \mathtt{L} & ::= & \Gamma \rhd \mathtt{P} \mid \Gamma \rhd \mathtt{D} \\
\text{Blocks} & \mathtt{k} & \in & \mathtt{K} & ::= & \emptyset_{\mathtt{K}} \mid \mathtt{k} \circ \mathtt{L} \mid \mathtt{K} \circ \{\mathtt{K}\}_{\vec{C}} \\
\text{Books} & \mathtt{b} & \in & \mathtt{B} & ::= & \emptyset_{\mathtt{B}} \mid \mathtt{B} \circ \mathtt{L} \mid \mathtt{B} \circ \{\mathtt{K}\}_{\vec{C}}
\end{array}$$

It starts with the line construction which could be seen as a step of reasoning. A line is a mathematical sentence expressed in a specific context. A context being a sequence of declarations ($\mathtt{Z}$) and assumptions (statements $\mathtt{P}$). Then we have blocks that group lines and sub-blocks together (expressing for example that several consecutive lines proving a certain proposition should be considered as one entity together). They allow one to specify a set of constants ($\{\ \}_{c_1,\ldots,c_n}$) which will be local to the block. We take $\vec{C}$ to be a list of C's. The use of these constants is then restricted to the block in question. $\emptyset_{\mathtt{K}}$ stands for the empty block. Lastly books are defined as sequences of lines and blocks. $\emptyset_{\mathtt{B}}$ stands for the empty book.

MathLang contexts are described using the three sets Γ, Γ_F and Γ_{FS}. $\emptyset_\Gamma$ stands for the empty context. To extend MV and WTT contexts, we have introduced in MathLang's abstract syntax a construction to scope variables or assumptions on several lines. This uses the flag notation already present in MV but only as syntax sugaring. Flags were used as syntax sugaring to avoid repetition of similar elements in consecutive contexts and so to reduce the size of MV or WTT examples. Moreover, flags help clarify the scope of a variable or an assumption over several lines. We introduce flags in MathLang's abstract syntax since we consider it important to have this scope information encoded in the language. Flags are composed of a head (a statement or a variable declaration in [ ]) and a flagstaff (several $\bullet$). We will use a specific notation for flags as shown here:

Normal notation		Flag notation
$[\mathtt{ez}_1], [\mathtt{ez}_2], \mathtt{ez}_3, \mathtt{ez}_4$	$\triangleright \mathtt{e}_1$	
$\bullet, \bullet, \mathtt{ez}_5$	$\triangleright \mathtt{e}_2$	
$\bullet, [\mathtt{ez}_6]$	$\triangleright \mathtt{e}_3$	
$\bullet, \bullet$	$\triangleright \mathtt{e}_4$	

$\mathtt{ez}_i$ is a statement or a variable declarations and $\mathtt{e}_i$ is a line body.

Example 2.3 Flags
Our MathLang translation of the definition and the proof of theorem 2 from chapter 1, section 2 illustrates the use of flags. In Figure 2, we give the original text of this example and an output of our MathLang translation. Note that line numbering starts at 24 since the earlier parts of the chapter occupy the other 23 lines. Note also that x' is the successor of x. We refer the reader to authors' web pages for the translation of all the first chapter.
By looking quickly at this example we see that a commonly used sentence "let a variable be something in the following ..." is easily expressible with a flag in MathLang. In this example the first two flags respectively introduce a set of variables and state that Theorem 2 holds for it. The third flag introduce the hypothesis that x is in $\mathfrak{M}$. With MathLang's flags, one could express that the variables x used in lines 27 to 29 stand for the same object. In MV and WTT these variables x would have been introduced three times (one time per line) and there would have been no possibility to retrieve the strong link that unify them together.

Example 2.4 Blocks
In MathLang we introduced in the abstract syntax the notion of blocks of lines. Blocks were already used at the metalevel in F3 of [Aut94] where they were linked with flags. In MathLang we have separated both notions: flags extend the contexts whereas blocks describe the structure of a text. In mathematics, sections or delimiters (of proofs or examples) give important information to the reader and help him understand and follow the author's reasonings. *Blocks* in MathLang describe the structure of the text. Our encodings above of *Definition 1* (see Example 2.2) and of the *Proof of Theorem 2* (see Figure 2 of Example 2.3), give examples of blocks.

Theorem 2

$$x' \neq x.$$

Proof. Let $\mathfrak{M}$ be the set of all x for which this holds true.

I) By Axiom 1 and Axiom 3,

$$1' \neq 1;$$

therefore 1 belongs to $\mathfrak{M}$.

II) If x belongs to $\mathfrak{M}$, then

$$x' \neq x,$$

and hence by Theorem 1,

$$(x')' \neq x',$$

so that x' belongs to $\mathfrak{M}$.

By Axiom 5, $\mathfrak{M}$ therefore contains all the natural numbers, i.e. we have for each x that

$$x' \neq x.$$

□

[Lan51]

$x : \mathbb{N} \rhd \mathrm{Th2}(x) := x \neq x'$ (24)

Proof Theorem 2 {2.2}

$\boxed{\mathfrak{M} : \mathrm{SET}}$

| $\boxed{\forall_{x:\mathfrak{M}} \mathrm{Th2}(x)}$

| | $\mathrm{Ax1}, \mathrm{Ax3}(1) \rhd 1' \neq 1$ (25)

| | $(25), (\mathrm{Def\ Th2}) \rhd 1 : \mathfrak{M}$ (26)

| | $\boxed{x : \mathfrak{M}}$

| | | $\rhd x' \neq x$ (27)

| | | $(27), \mathrm{Th1}(x', x) \rhd x'' \neq x'$ (28)

| | | $(28), (\mathrm{Def\ Th2}) \rhd x' : \mathfrak{M}$ (29)

| | $\mathrm{Ax5}(\mathfrak{M}, (26), (29)) \rhd \mathbb{N} \subset \mathfrak{M}$ (30)

$(30) \rhd \forall_{x:\mathbb{N}} \mathrm{Th2}(x)$ (31)

Fig. 2. Flags example

In MathLang, a block denotes a structure. It is possible to restrict the definition of a constant to a precise block, we call this constant *local*. An example of these local constant could be seen in our translation of the first chapter of [Lan51] with blocks {2.5.1} and {4.2.1}. Each of these blocks represents a part of a proof of a theorem (respectively part A of Theorem 4 and part A of Theorem 28) where the local constants a and b are defined.

Although we make heavy use of blocks and we fully incorporate them in the implementation, we still do not have rules that help derive useful information about these blocks. For example, we still cannot formally state that a certain block is a proof of a given theorem.

Example 2.5 References
The example given in Figure 3, from chapter 1, section 2 (part of the proof of Theorem 4), illustrates the use of line and definition references in MathLang. Referencing is already implemented but is not yet part of the definition of MathLang's abstract syntax. We have annotated the original text with the line numbers of our translation in parentheses.

Let x belong to $\mathfrak{M}$, so that there exists an $x+y$ for all y. Then the number

$$x' + y = (x+y)'$$

is the required number for x', since

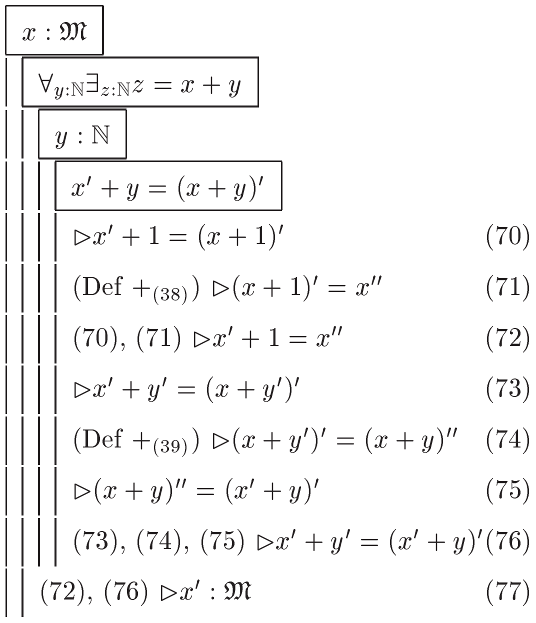
$$x' + 1 =_{(70)} (x+1)' =_{(71)} (x')' \quad (72)$$

and

$$\begin{aligned} x' + y' &=_{(73)} (x+y')' \\ &=_{(74)} ((x+y)')' \\ &=_{(75)} (x'+y)'. \qquad (76) \end{aligned}$$

Hence x' belongs to $\mathfrak{M}$. (77)

[Lan51]



$\boxed{x : \mathfrak{M}}$

$\boxed{\forall_{y:\mathbb{N}}\exists_{z:\mathbb{N}} z = x+y}$

$\boxed{y : \mathbb{N}}$

$\boxed{x' + y = (x+y)'}$

$\rhd x' + 1 = (x+1)'$ (70)

(Def $+_{(38)}$) $\rhd (x+1)' = x''$ (71)

(70), (71) $\rhd x' + 1 = x''$ (72)

$\rhd x' + y' = (x+y')'$ (73)

(Def $+_{(39)}$) $\rhd (x+y')' = (x+y)''$ (74)

$\rhd (x+y)'' = (x'+y)'$ (75)

(73), (74), (75) $\rhd x' + y' = (x'+y)'$ (76)

(72), (76) $\rhd x' : \mathfrak{M}$ (77)

Fig. 3. References example

References to previous steps of the reasoning are commonly used in mathematics. A sentence like "By definition of ..." could be encoded in MathLang by adding in the line context a reference to the definition of a constant.

In addition to such referencing, mathematicians use the habit of grouping several steps of reasoning together. For example, to show that $x'+1 = (x')'$, E. Landau first states that $x' + 1 = (x+1)'$ and then states that $(x+1)' = (x')'$ and writes these steps in one equation. In MathLang we have represented $A = B = C$ as three explicit steps of reasoning. Each step corresponds to one line. The first one states that $A = B$. The second states that $B = C$. Then the third concludes that $A = C$ by referencing the two previous lines. References make this kind of reasoning constructions easier to encode.

3 The derivation rules of MathLang

The derivation rules of MathLang assign (unique) types to well formed MathLang expression. The set of types is a subset of the grammatical categories. We list here all the types which are atomic: `T`, `S`, `N`, `A`, `P`, `D`, Γ, `K`, `B`. In the example given in Figure 1, the type analysis assigns types corresponding to the grammatical categories while the sentence is well formed. The same sentence slightly modified will this time be badly formed. Figure 4 points out the type error.

The type analysis will raise an error while applying the LINE-IN-BOOK (or LINE-IN-BLOCK) rule. The context is valid but the expression given as the body of line $(x+y)$ should either be a statement (`P`) or a definition (`D`) while here

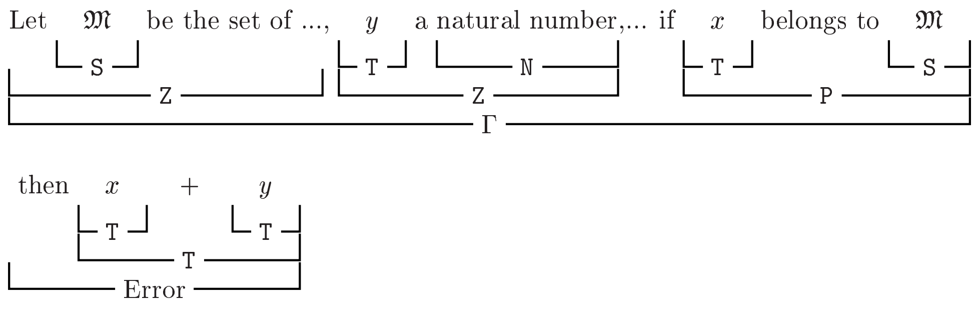

Let 𝔐 be the set of ..., y a natural number,... if x belongs to 𝔐
S
T
N
T
S
Z
Z
P
Γ
then x + y
T
T
T
Error

Fig. 4. An invalid mathematical line and its assigned MathLang types

$x + y$ is a term (T). See later on in this Section for a description of the derivation rules.

Typing notations.

Before giving the derivation rules of MathLang we need to make precise the notations we use. A *typing* states that a *type* is assigned to an *expression* in a certain *environment*. This is written as follow: *environment* $\vdash$ *expression* **:** *type*. The *environment* consists of a book in which the expression is typed. For expressions at the sentence and phrase level, the *environment* also contains a context.

Flattening flags.

The flag construction we have introduced in the previous section has an important role in the encoding of mathematical texts. To make our typing rules more readable these are only applicable to MathLang without flags (context elements are restricted to assumptions from P and declarations from Z). We are able by *flattening* each flag of a MathLang text to keep the typing information they contained. Each flagstaff element is then replaced by the content of the corresponding flag's head. With this flattening operation we lose the information that the same assumption holds on several lines or that a variable stands for the same object on several lines as well, but this does not change the typing of a book. The derivation rules are defined on MathLang texts obtained after doing a flags flattening operation.

Functions.

We use in the set of derivation rules some specific functions. We only describe these functions here rather than giving their full formal definitions.

- `dvars` returns the set of variables declared in a given context ($\texttt{dvars} : \Gamma \to \wp(V)$).
- $\texttt{dcons}_{\texttt{B}}$ returns the set of defined constant in a given book. This excludes local constants ($\texttt{dcons}_{\texttt{B}} : \texttt{B} \to \wp(C)$).

- $\mathtt{dcons}_{\mathtt{K}}$ returns the set of defined constant in a given block. This excludes constants defined locally in inner blocks ($\mathtt{dcons}_{\mathtt{K}} : \mathtt{K} \to \wp(C)$).
- $\mathtt{in}_C(i, c, \mathtt{b})$ gives the type of the i^{th} argument of constant c as defined in the book $\mathtt{b}$.
- $\mathtt{in}_B(i, b, \mathtt{b})$ represents the type expected by the binder b.
- $\mathtt{fvars}$ returns the set of free variables of a given expression ($\mathtt{fvars} : \mathtt{E} \to \wp(V)$).
- $\mathtt{casedcons}_{\mathtt{B}}$ (resp. $\mathtt{casedcons}_{\mathtt{K}}$) returns the set of constants defined by cases in a given book (resp. block), excluding local definitions ($\mathtt{casedcons}_{\mathtt{B}} : \mathtt{B} \to \wp(C)$ and $\mathtt{casedcons}_{\mathtt{K}} : \mathtt{K} \to \wp(C)$).
- $\mathrm{OK}(\mathtt{b}\,;\,\gamma)$ is an abbreviation for $\vdash \mathtt{b} \mathrel{:} \mathtt{B}$ and $\mathtt{b} \vdash \gamma \mathrel{:} \Gamma$.

Rules for expressions.

These rules assign a type to the expressions taken from the phrase and the sentence levels. That is to say variable instances, constant calls, binding expressions and adjective attributions. A special rule REC-CONS allows calls to annotated constant which is currently being defined (c' with its input types wt_i). This rule allows recursive definitions of constants (see rules FULL-DEF and CASE-DEF-FIRST).

$$\frac{\mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad v \in V^{\mathtt{T/S/P}} \qquad v \in \mathtt{dvars}(\gamma)}{\mathtt{b}\,;\,\gamma \vdash v \mathrel{:} \mathtt{T/S/P}}\ \textsc{var}$$

$$\frac{\begin{array}{c}\mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad c \in C^{\mathtt{T/S/N/A/P}} \\ c \in \mathtt{dcons}_{\mathtt{B}}(\mathtt{b}) \qquad \forall i \in \{1,\ldots,n\},\ \mathtt{b}\,;\,\gamma \vdash \mathtt{e}_i \mathrel{:} \mathtt{in}_C(i, c, \mathtt{b})\end{array}}{\mathtt{b}\,;\,\gamma \vdash c(\mathtt{e}_1,\ldots,\mathtt{e}_n) \mathrel{:} \mathtt{T/S/N/A/P}}\ \textsc{cons}$$

$$\frac{\begin{array}{c}\mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad c \in C^{\mathtt{T/S/N/A/P}} \\ c = c' \qquad \forall i \in \{1,\ldots,n\},\ \mathtt{b}^{\mathtt{REC}^{c'}_{wt_1,\ldots,wt_n}}\,;\,\gamma \vdash \mathtt{e}_i \mathrel{:} wt_i\end{array}}{\mathtt{b}^{\mathtt{REC}^{c'}_{wt_1,\ldots,wt_n}}\,;\,\gamma \vdash c(\mathtt{e}_1,\ldots,\mathtt{e}_n) \mathrel{:} \mathtt{T/S/N/A/P}}\ \textsc{rec-cons}$$

$$\frac{\mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad b \in B^{\mathtt{T/S/N/A/P}} \qquad \mathtt{b}\,;\,\gamma,\mathtt{z} \vdash \mathtt{e} \mathrel{:} \mathtt{in}_B(b)}{\mathtt{b}\,;\,\gamma \vdash b_{\mathtt{z}}(\mathtt{e}) \mathrel{:} \mathtt{T/S/N/A/P}}\ \textsc{bind}$$

$$\frac{\mathtt{b}\,;\,\gamma \vdash \mathtt{e}_1 \mathrel{:} \mathtt{A} \qquad \mathtt{b}\,;\,\gamma \vdash \mathtt{e}_2 \mathrel{:} \mathtt{N}}{\mathtt{b}\,;\,\gamma \vdash \mathtt{e}_1\mathtt{e}_2 \mathrel{:} \mathtt{N}}\ \textsc{attr}$$

Grouping rules. In the above rules, we use the symbol / to group similar rules. For example the VAR rule is in place of three rules, the first of which is:

$$\frac{\mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad v \in V^{\mathtt{T}} \qquad v \in \mathtt{dvars}(\gamma)}{\mathtt{b}\,;\,\gamma \vdash v \mathrel{:} \mathtt{T}}\ \textsc{var-term}$$

Typically, the rule VAR assigns the type corresponding to the variable's grammatical category. The constant rules CONS and REC-CONS will do the same after checking the coherence of the arguments' typings with what the constant expects (the function $\mathtt{in}_C$ gives the list of types of the constant's parameters). The BIND rule, in addition, introduces the new variable in the typing environment of the inner expression. The rule ATTR describes how to construct a new noun by attributing an adjective to an existing noun.

Rules for contexts.

These rules check the coherence of contexts. The first one states that an empty context is a valid context. The last one checks if an assumption in the context is a well formed statement expression. The three remaining rules correspond to the three constructions of variable introduction. The grammatical category of the introduced variable must fit the kind of construction used: either a set or a statement or a term.

$$\frac{\vdash \mathtt{b} \boldsymbol{:} \mathtt{B}}{\mathtt{b} \vdash \emptyset_\Gamma \boldsymbol{:} \Gamma}\ \textsc{empty-cont}$$

$$\frac{\mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad v \in V^{\mathtt{S}} \qquad v \notin \mathtt{dvars}(\gamma)}{\mathtt{b} \vdash \gamma, v : \mathrm{SET} \boldsymbol{:} \Gamma}\ \textsc{set-var-decl}$$

$$\frac{\mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad v \in V^{\mathtt{P}} \qquad v \notin \mathtt{dvars}(\gamma)}{\mathtt{b} \vdash \gamma, v : \mathrm{STAT} \boldsymbol{:} \Gamma}\ \textsc{stat-var-decl}$$

$$\frac{\mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad v \in V^{\mathtt{T}} \qquad v \notin \mathtt{dvars}(\gamma) \qquad \mathtt{b}\,;\,\gamma \vdash \mathtt{e} \boldsymbol{:} \mathtt{S/N}}{\mathtt{b} \vdash \gamma, v : \mathtt{e} \boldsymbol{:} \Gamma}\ \textsc{term-var-decl}$$

$$\frac{\mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad \mathtt{b}\,;\,\gamma \vdash \mathtt{p} \boldsymbol{:} \mathtt{P}}{\mathtt{b} \vdash \gamma, \mathtt{p} \boldsymbol{:} \Gamma}\ \textsc{assump}$$

Rules for definitions.

There are two kinds of definitions in the abstract syntax with three typing rules. First, we have a rule for the basic constant definition which provides a unique expression as a value for the constant. Then there are two other rules for the definitions by cases: one for the first (basic) case, the other for the following cases and checks if the types of the constant's arguments correspond to those of the first case.

$$\frac{\begin{array}{c} \mathrm{OK}(\mathtt{b}\,;\,\gamma) \\ c \in C^{\mathtt{T/S/N/A/P}} \qquad c \notin \mathtt{dcons}_{\mathtt{B}}(\mathtt{b}) \qquad \mathtt{dvars}(\gamma) = \cup_{i=1}^{n} v_i \\ \forall i \in \{1,\ldots,n\},\ v_i \in V^{wt_i} \qquad \mathtt{b}^{\mathtt{REC}^{c}_{wt_1,\ldots,wt_n}}\,;\,\gamma \vdash \mathtt{e} : \mathtt{T/S/N/A/P} \end{array}}{\mathtt{b}\,;\,\gamma \vdash c(v_1,\ldots,v_n) := \mathtt{e} : \mathtt{D}} \ \textsc{full-def}$$

$$\frac{\begin{array}{c} \mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad c \in C^{\mathtt{T/S/N/A/P}} \\ c \notin \mathtt{casedcons}_{\mathtt{B}}(\mathtt{b}) \qquad \mathtt{dvars}(\gamma) = \cup_{i=1}^{n} \mathtt{fvars}(\mathtt{e}_i) \\ \forall i \in \{1,\ldots,n\},\ \mathtt{b}\,;\,\gamma \vdash \mathtt{e}_i : wt_i \\ \mathtt{b}^{\mathtt{REC}^{c}_{wt_1,\ldots,wt_n}}\,;\,\gamma \vdash \mathtt{e} : \mathtt{T/S/N/A/P} \end{array}}{\mathtt{b}\,;\,\gamma \vdash c(\mathtt{e}_1,\ldots,\mathtt{e}_n) := \mathtt{e} : \mathtt{D}} \ \textsc{case-def-first}$$

$$\frac{\begin{array}{c} \mathrm{OK}(\mathtt{b}\,;\,\gamma) \qquad c \in C^{\mathtt{T/S/N/A/P}} \\ c \in \mathtt{casedcons}_{\mathtt{B}}(\mathtt{b}) \qquad \mathtt{dvars}(\gamma) = \cup_{i=1}^{n} \mathtt{fvars}(\mathtt{e}_i) \\ \forall i \in \{1,\ldots,n\},\ \mathtt{b}\,;\,\gamma \vdash \mathtt{e}_i : \mathtt{in}_C(i,c,\mathtt{b}) \\ \mathtt{b}\,;\,\gamma \vdash \mathtt{e} : \mathtt{T/S/N/A/P} \end{array}}{\mathtt{b}\,;\,\gamma \vdash c(\mathtt{e}_1,\ldots,\mathtt{e}_n) := \mathtt{e} : \mathtt{D}} \ \textsc{case-def-alter}$$

In rules full-def and case-def-first, to allow recursive definitions, we are annotating the environment (notation $\mathtt{b}^{\mathtt{REC}^{c}_{wt_1,\ldots,wt_n}}$) in which we will type the body of the definition (expression $\mathtt{e}$). This annotation contains the constant currently defined and the types of its parameters. This information will be kept in the environment to allow calls to this constant (see rule rec-cons).

Rules for blocks.

A block could either be empty, end with a line or end with a block. The rules below describe how to build a block from an already well-formed one. This is done by adding a line or a sub-block to the existing block.

$$\frac{\vdash \mathtt{b} : \mathtt{B}}{\mathtt{b} \vdash \emptyset_{\mathtt{K}} : \mathtt{K}} \ \textsc{empty-block}$$

$$\frac{\mathtt{b} \vdash \mathtt{k} : \mathtt{K} \qquad \mathtt{b} \circ \{\mathtt{k}\}_{\emptyset}\,;\,\gamma \vdash \mathtt{p/d} : \mathtt{P/D}}{\mathtt{b} \vdash \mathtt{k} \circ \gamma \rhd \mathtt{p/d} : \mathtt{K}} \ \textsc{line-in-block}$$

$$\frac{\begin{array}{c} \vdash \mathtt{b} : \mathtt{B} \\ \mathtt{b} \vdash \mathtt{k} : \mathtt{K} \qquad \{c_1,\ldots,c_n\} \subseteq \mathtt{dcons}_{\mathtt{K}}(\mathtt{k}') \qquad \mathtt{b} \circ \{\mathtt{k}\}_{\emptyset} \vdash \mathtt{k}' : \mathtt{K} \end{array}}{\mathtt{b} \vdash \mathtt{k} \circ \{\mathtt{k}'\}_{c_1,\ldots,c_n} : \mathtt{K}} \ \textsc{block-in-block}$$

Rules for books.

The rules for books are similar to the rules for blocks while a book could be seen as the outermost block.

$$\frac{}{\vdash \emptyset_{\texttt{B}} : \texttt{B}}\ \textsc{empty-book} \qquad \frac{\vdash \texttt{b} : \texttt{B} \qquad \texttt{b}\,;\gamma \vdash \texttt{p/d} : \texttt{P/D}}{\vdash \texttt{b} \circ \gamma \rhd \texttt{p/d} : \texttt{B}}\ \textsc{line-in-book}$$

$$\frac{\vdash \texttt{b} : \texttt{B} \qquad \{c_1, \ldots, c_n\} \subseteq \texttt{dcons}_{\texttt{K}}(\texttt{k}) \qquad \texttt{b} \vdash \texttt{k} : \texttt{K}}{\vdash \texttt{b} \circ \{\texttt{k}\}_{c_1,\ldots,c_n} : \texttt{B}}\ \textsc{block-in-book}$$

4 Implementation

A main improvement of MathLang over MV and WTT is the implementation we have made of a language checker which was fully guided by the translation of the first chapter.

Concrete syntax

Our choice for the concrete syntax was to use XML recommendations. We intend this XML syntax to be used only by the MathLang framework and by software developers. The MathLang user, instead of using a concrete syntax, will use a specific user-friendly editor (see Section 5). This choice has been made to fit in our main requirement for a mathematical software which is to be easy to use by mathematicians. By choosing a computer oriented syntax we avoid the need of the user to learn a new specific syntax. This is avoided by including MathLang in a common scientific text editor.

In MathLang's concrete syntax, each occurrence of an identifier is composed by its name and the grammatical category of which it is part. The concrete syntax is then mainly verbose. This facilitates the checking and allows local analysis of the typing. This duplication of information also eases the design of software rendering and enforces the use of an editor rather than allowing editing by hand.

The type checker

We have implemented a type checker that analyses a MathLang text by applying the rules of the type system (see Section 3). This software has been programmed using Camlp4 (parser part) and OCaml (type inference part). This program checks the typing of a given MathLang document. If the typing succeeds, the XML document is then considered as a well formed MathLang book. Otherwise, if the document is not correctly formed, the checker will point the syntax or type error that has been found. We have used our checker during the translation of Landau's first chapter.

Rendering

We apply syntactic transformations on MathLang texts in order to obtain presentable versions of the XML concrete syntax. For this paper we have used one XSL transformation that produces a LaTeX document. We have automatically generated the examples of Section 2.2 and the full translation of the first chapter of [Lan51] using this transformation. This rendering is experimental. It remains close to MV and WTT while also being clearer thanks to the flags and blocks, and also to some syntactic sugaring (the grammatical category of each identifier is not printed and we use common symbols to help the reader). We explain our goals for MathLang rendering in Section 5.

Translation

In order to experiment with our language MathLang we have used in its development, a translation of E. Landau's *Foundations of Analysis*[Lan51]. Translating an existing mathematical text and checking the resulting MathLang text with our software show the *feasibility* of such a mathematical encoding. This experience demonstrates the *usability* of our language and guides our research and development as closely as possible to effective *mathematicians' writing habits*. We have achieved so far the translation of the full first chapter. This translation and the original text can be found in an appendix to this article reachable at the authors' web pages.

5 Related and Future work

M. Scheffer's master thesis [Sch03] gives a translation of the same first chapter of [Lan51] but in the original calculus WTT with sugared notation called WTT_S. The meta theory developed for WTT did not involve the syntactic sugaring and we suspect that R. Körvers' implementation of WTT (upon which M. Scheffer's encoding was based) did not involve the syntactic sugaring either. This means that:

- There is a gap between the language used in M. Scheffer's encoding and the theory and its implementation. Our encoding on the other hand takes place in a framework where there is a harmony between the language, its metatheory and its implementation. We give the translation in MathLang, an extended version of WTT where much of the sugaring of WTT is replaced by first class status in MathLang. Furthermore, the implementation is that of MathLang and hence we can guarantee the coherence of our encoded texts. This would be hard to guarantee in M. Scheffer's case since the implementation is of WTT whereas the encoding is in another language WTT_S.
- The rendering we automatically obtain from the encoding is closer to the mathematicians' text than that of Scheffer's. This is since both the language MathLang and its implementation contain flags and blocks.
- Moreover, references to lines, definitions and blocks make our encoding closer to the mathematician's intentions.

As we said earlier, MathLang is still under development. We believe that it is important that this development be guided by existing mathematical texts and practice and that the encoding should be in the language under development and not in syntactic sugaring versions of it. It is also important that the metatheory and implementation be as faithful as possible to the language itself.

M. Scheffer also discusses the step of moving from WTT into a more formal language. Although this step is an important future direction, we believe that first, a more stable language of mathematics needs to be found and extensively tested before moving into higher levels of formalisation. Already, we have developed many type theories with many of the features of Automath (e.g., explicit substitutions, parameters, definitions, local and/or global reductions, notions of unification). These type theories will guide us into the more formal levels of MathLang. But, to avoid the danger of creating "toy" levels of formalisation, we need to have a good basis for the mathematical language under development which can then lead us into more trustworthy formal levels.

There exist different frameworks for putting mathematics on the computer, each with a particular aim: calculation, analysis, storage, visualization, checking, etc. However, none of these frameworks is expressive enough to allow the integration of many aims in one system. In [KW01], we proposed to design a language influenced by MV [Aut94] and WTT [KN04] and driven by the encoding of three books: 1) the *Landau's Foundations of Analysis* [Lan51] since it was fully-formalised in Automath (and hence provides an excellent basis for comparison), 2) the *Elements of Euclid* [Hea56] since it is known to have many errors (and hence helps distinguish between logical and structural correctness), and 3) The *Compendium of Lattices* [GHK$^+$80] since 60% of it has been formalised in Mizar. We have not yet tackled the third book. On the other hand, work on Euclid's book is under development by a student for his degree project while work on Landau's book has been in progress for over a year now [Maa03], and we have already completed the translation of the first chapter while building at the same time, a basis of a refined language MathLang, its implementation and useful associated software packages. The immediate future work on MathLang and its framework will follow three directions strongly linked together.

- **Specifications.** The MathLang language is still in development. We need to incorporate references in the MathLang syntax and to put the notion of blocks into action by adding some constructions to enable us to derive information about blocks. These extensions will always be made in parallel with the guiding translation work. At every stage, we will make sure that the metatheory is well developed and that the implementation is faithful to the language.
- **Translation.** Continuing the translation of Landau's book (and making progress on the translation of Euclid's book) is a main target to guide the

design of MathLang. This translation guides us as to how we should further develop MathLang and enables us to build an implementation that is easy to use which we can then pass for external feedbacks from mathematicians or programmers as our future users.

- **Framework.** Because our language will be as exhaustive as possible in its way to encode mathematical texts we will need to have a specific editor to assist the mathematician. This editor should have a user friendly interface. It should be able to print mathematical symbols on the screen. An integrated checker should give instantaneous feedback about the type analysis of the text. We have planned and started to use the What-You-See-Is-What-You-Get editor TeXmacs [2] for this development. TeXmacs is an editor dedicated to mathematical texts. It is mostly interactive and has been developed to allow extensions. MathLang's framework should assist mathematicians to create new documents and to translate existing mathematical texts (such as LaTeX and OMDoc documents). For this second purpose we aim to develop a tool to semi-automate the work of translation. The work described in the article [BS03] is a good basis for this development. Y. Baba and M. Suzuki present a grammatical analyser which extends its own grammatical rules on the fly. After translating one sentence, the grammatical analyser will be able to recognize the same kind of pattern used by the mathematician in the rest of the text.

In addition to this discussed development of a language of mathematics MathLang, we will also use earlier developments we made on extensions of type systems (with notions like definitions, explicit substitutions, parameters and higher order unification) in order to take our full encoding of Landau's book in MathLang into a fully formalised version which can then be compared with the existing Automath formalisation. At each stage, we will be working on general methods that can be applied to other books.

References

[MKM03] Asperti, Buchberger and Davenport (Eds.). *The Second International Conference on Mathematical Knowledge Management, MKM 2003*, Bertinoro, Italy, February 16-18. Lecture Notes in Computer Science 2594, 2003.

[BS03] Yusuke Baba and Masakazu Suzuki. An annotated corpus and a grammar model of theorem description. In [MKM03], 2003.

[GHK$^+$80] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *A Compendium of Continuous Lattices.* Springer-Verlag, 1980.

[Hea56] Heath. *The 13 Books of Euclid's Elements.* Dover, 1956.

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

[KN04] Fairouz Kamareddine and Rob Nederpelt. A refinement of de Bruijn's formal language of mathematics. To appear in Journal of Logic, Language and Information. 2004.

[Koh03] Michael Kohlhase. *OMDoc: An Open Markup Format for Mathematical Documents (Version 1.1)*. http://www.mathweb.org/omdoc, 2003.

[KW01] Fairouz Kamareddine and Joe Wells. Promath: presenting, proving and programming mathematical books. Revised under the title: MathLang, A new language for Mathematics, Logic and Computation, August 2001.

[Lan30] Edmund Landau. *Grundlagen der Analysis*. Chelsea, 1930.

[Lan51] Edmund Landau. *Foundations of Analysis*. Chelsea, 1951. Translation of [Lan30] by F. Steinhardt.

[Maa03] Manuel Maarek. First year PhD report. Technical report, Heriot-Watt University, August 2003.

[Aut94] R.P. Nederpelt and J.H. Geuvers and R.C. de Vrijer (Eds.), *Selected papers on Automath*, North-Holland, 1994.

[Sch03] Mark Scheffer. Formalizing Mathematics using Weak Type Theory. Master's thesis, Technische Universiteit Eindhoven, September 2003.

[vBJ77] L. S. van Benthem Jutting. *Checking Landau's "Grundlagen" in the AUTOMATH system*. PhD thesis, Eindhoven University of Technology, 1977. Mathematical Centre Tracts nr. 83, Math. Centre, Amsterdam 1979.

[Wie03] Freek Wiedijk. Comparing mathematical provers. In [MKM03], 2003.