MathAbs: A Representational Language for Mathematics

Muhammad Humayoun and Christophe Raffalli
Laboratory of Mathematics (LAMA), Université de Savoie, France.
muhammad.humayoun@etu.univ-savoie.fr, christophe.raffalli@univ-savoie.fr

ABSTRACT

MathAbs (**Math**ematical **Abs**tract language) is a system independent formal language for mathematical texts that can preserve its structure including the line of reasoning, proof steps and logical structure. By system independent, we mean that its formalism is not based on any specific logic, theory or proof assistant.

MathAbs is intended only for machine manipulation and it is part of a system called MathNat[6], which provides a controlled language to write mathematical texts found in textbooks and published work. Here, MathAbs is used as an intermediary between the natural language of the mathematician and the formal language of the logician. **We give here a detailed account of MathAbs, its justification, formal definition and semantics.**

1. BACKGROUND AND CONTEXT

The language of mathematics[1] mainly consists of natural language (NL), symbolic expressions and notations which evolved in centuries. It is flexible, structured and semantically well-understood by mathematicians. By the virtue of Gödel completeness theorem[4], it is generally accepted that mathematics could be formalized; for instance in first-order logic or in set theory; using Hilberts' system, natural deduction or sequent calculus for proofs.

However it is very difficult to formalize mathematical texts automatically. Some of the reasons for this enterprise are: (1) the texts may contain some complex and rich linguistic features of NL such as the use of anaphoric pronouns and references, rephrasing, distributive vs. collective readings, etc; (2) the texts could be inherently ambiguous if not written carefully; and (3) mathematical proofs may contain reasoning gaps, which are hard to fill using automated theorem provers.

[1]By the language of mathematics, we mean prose that mathematicians use to author textbooks and lecture notes. We will refer them as 'mathematical texts'.

FIT'10 , December 21–23, 2010, Islamabad, Pakistan.

These problems convince a multitude of logicians and scientists that if not impossible, automatic formalization of the mathematics in its textual form is far more difficult and problematic. For instance, on the use of the language of mathematics, a famous logician, Frege expressed his dissatisfaction in following words ([3], Preface):
"*... I found the inadequacy of language to be an obstacle; no matter how unwieldy the expressions was ready to accept, I was less and less able, as the relations became more and more complex, to attain the precision that my purpose required.*"

This leads to the current state of art of theorem proving, in which the language of mathematics is given up and mathematical texts are sometimes manually formalized in very precise and accurate systems using specific formalisms normally based on some particular calculus or logic. Isar[12] for Isabelle, the notion of *formal proof sketches*[13] for Mizar and Mathematical Proof Language *MPL*[1] for Coq are some recent examples. Such a formal piece of mathematics does not contain natural language elements[2] at all. Instead, such languages are quite restricted and non ambiguous having a programming language like syntax, with a few syntactic constructions. Therefore, the mathematician cannot write the mathematical texts in a narrative style as he does in published material using these languages.

So we can safely say that no formalism and logical system is able to replace the language of mathematics for a mathematician. The above discussion leads to the following question:
Can we build a program that automatically formalizes mathematical texts and can we mechanically verify their correctness?
Answering this question positively requires the handling of two very hard problems: (1) parsing the mathematical texts mainly proofs and translating those parse trees to a formal language (in our case MathAbs) after resolving linguistic issues; and (2) validation of the formal version of mathematical texts.

MathNat[3] (**Math**ematics in controlled **Nat**ural language) project aims at being a first step towards answering this gigantic question. It is an on going project which is still a *proof-of-concept*; see [6] for more details. MathNat has three major components:

1. **The Controlled Language of Mathematics (CLM):** It is a computer processable subset of English and symbolic language for writing mathematics. It is controlled yet remarkably closer to the mathematical texts. It

[2]The use of natural language and its narrative style, etc
[3]Homepage: `www.lama.univ-savoie.fr/~humayoun/phd/mathnat.html`

supports numerous linguistic features such as anaphoric pronouns and references, rephrasing of a sentence in multiple ways producing canonical forms and the proper handling of distributive and collective readings. Furthermore, using CLM does not presuppose any expertise in formal logic or computational linguistics.

2. **Automatic formalization of CLM to MathAbs:** For formalization, the host system of MathNat automatically translates CLM into MathAbs, where all ambiguities of natural language have been resolved. This paper is mainly devoted to a precise description of MathAbs.
3. **Validation:** We give a precise semantics to MathAbs in §4, allowing for a clear definition of its validation. In the current implementation, MathAbs is translated to first-order formulas and validated by automated theorem prover. Other approaches, such as using proof assistants (Coq, HOL, etc) could be interesting but have not been explored yet.

The aim of this paper is to give a detailed account of MathAbs. We give its motivation in §2, syntax in §3, formal definition and semantics in §4 and completeness in §5. As a running example, a typical mathematical text and its MathAbs version is given in figure 1. Two more examples could be found in Appendix. All of them are written in the controlled language supported by MathNat.

2. WHY MATHABS

The use of an intermediate language between the language of mathematics and theorem provers is not new. AutoMath[2] of N.d. Bruijn, is one of the pioneering such work in which a very restricted proof language was proposed. Similarly, a recent framework MathLang[7], proposes an intermediate language named Core Grammatical aspect (CGa) which is a predecessor of Weak Type Theory (WTT) and similar to AutoMath. MathLang supports the manual annotation of mathematical texts, instead of automatic formalization by parsing (as MathNat does). Once the annotation is done by the author, a number of transformations to the annotated text is automatically performed for automatic verification in Mizar and Isabelle.

The intermediate languages such as AutoMath, CGa, WTT, etc[4] do not fit our needs because: (1) they are low level and the content representation is not close enough to the informal proofs; (2) they are mostly good for the computerization of mathematics but not formalization. By computerization we mean the process of putting a document in a computer format for knowledge management ([8]:32)[5].

Like these languages, MathAbs acts as a mediator between the language of mathematics and the proof checking systems in MathNat system. However, unlike these languages, MathAbs has a clear semantics for axioms, definitions, theorems and proofs. MathAbs follows arbitrary rules of reasoning, having no dependence to any particular theory or logical system. It attempts to faithfully represent the language of mathematics which may have reasoning gaps. By "faithfully", we mean that the formal version of mathematics is as close as possible to the intentions expressed in the mathematical text. This language is intended to be simpler and non-ambiguous than the language of mathematics found in the published literature, so that we can interpret it strictly.

The proof language of MathAbs (which is its main aspect) is constructed by looking at proofs in natural language and observing that mathematicians do not give the name of the logical rules they use (except for induction and absurdity reasoning), but rather explain how the *context* is evolving. That is, what are the new hypotheses, the current goals; and some justifications such as the use of definitions of previous results.

It is important to note that a wrong proof can still be represented in MathAbs. It is so because MathAbs is only a representation language; so no validation or proof checking is performed directly in MathAbs as opposite to proof systems like natural deduction.

2.1 Origin of MathAbs

MathAbs is an extension of a proof language: new_command. It was designed as an intermediate formalism between natural language proofs and the proof assistant PhoX[9] in the DemoNat project[6][10, 11]. In DemoNat, the work was mainly focused on the development of a proof assistant that can utilize justifications found in informal proofs to reduce it's search space. By "justifications", we mean explanations or details whose purpose is to guide a reader about reasoning steps. Although an automatic translation from informal proofs to new_command was proposed but it has never been implemented. In this sense, MathNat is the successor of DemoNat.

The proof language new_command includes some of the standard commands of PhoX and support a proposition (formula) as a theorem statement. We adopt it after removing such dependencies and simplifying the proof language by reducing the constructs into minimal. Further, we add a language of definition and theorem. We also provide the semantics of MathAbs for axioms, definitions, theorems and their proofs, which was not done before.

3. SYNTAX OF MATHABS

MathAbs can represent theorems and their proofs along with supporting axioms and definitions. On a macro level, a MathAbs' document is a sequence of definitions, axioms and theorems with their proofs. However, analogous to the language of mathematics, the most significant part of MathAbs is the language of proof; while the definitions and axioms are probably just a fraction.

A proof in MathAbs is described as a tree of logical (meta) rules. These rules are arbitrary and do not necessarily have to be the rules of natural deduction or sequent calculus. Intuitively, at each proof step, there is an implicit active sequent with some hypotheses and one conclusion, which is being proved, and some other sequents to be proved later. The math text explains how the active sequent is modified to progress in the proof and some justifications (hints) may be given. A theorem forms the initial sequent with some hypotheses and one goal, which is then passed to the proof. While axioms and definitions are added in the initial sequent as hypotheses. The proof in figure 1 is a shortened version (we have merged some rules) of the semantics of the proof

[4]Similarly semantic markup languages such as ActiveMath, OpenMath, OMDoc, etc also do not fit our needs as they don't have a language of proof and its semantics.

[5]This reference only points to MathLang, but we find it true for all intermediate languages reported here.

[6]http://wiki.loria.fr/wiki/Demonat

1. **Definition 1 (divisibility).** Let m and n be arbitrary integers with a condition that $m > 0$. Then n is said to be divisible by m if there is a number q, such that $n = q * m$.
2. **Definition 2 (evenness).** An integer n is even if it is divisible by 2.
[. . .]
3. **Theorem.** If x and y are two even integers then $x+y$ is even.
4. **Proof.** By the definition of even numbers, we assume that $x + y = 2 * a + 2 * b$, where a and b are integers.
5. We deduce that $x + y = 2 * (a + b)$ by the last statement.
6. Therefore, $x + y$ is an even integer because it is a multiple of 2.

(a)

1. Definition 1 (divisibility). let m, n : `Integer` assume $m > 0$ define $divisible(n, m)$ as $\exists q$: `Number`$(n = q * m)$
2. Definition 2 (evenness). let n : `Integer` define $even(n)$ as $divisible(n, 2)$
[. . .]
3. Theorem 1. let x, y : `Integer` assume $even(x) \wedge even(y)$ show $even(x + y)\bullet$;
4. Proof. let a, b : `Integer` assume $x + y = 2 * a + 2 * b$ by def `Even_Number`$\bullet$
5. deduce $x + y = 2 * (a + b)$ by form $x + y = 2 * a + 2 * b\bullet$
6. deduce $multiple\_of([x + y], 2)\dagger$ show $even(x + y)$ by form $multiple\_of([x + y], 2)\bullet$ trivial

(b)

†in line 6 in (b), first, we deduce the justification i.e. $multiple\_of([x + y], 2)$, because it is not in the list of available hypothesis and then deduce the whole statement. However in line 5 in (b), we do not deduce the justification $(x + y = 2 * a + 2 * b)$ because it is already in the list of available hypothesis.

Figure 1: A typical text from elementary number theory and its MathAbs

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_2 \vdash x + y = 2 * (a + b)}\ 5.}{\dfrac{}{\Gamma_3 \vdash multiple\_of([x + y], 2)}\ 6.} \quad \dfrac{}{\Gamma_4 := (\Gamma_3, multiple\_of([x + y], 2)) \vdash even(x + y)}\ 6.\ \textit{show, trivial}}{\Gamma_3 := (\Gamma_2, x + y = 2 * (a + b) \vdash even(x + y)}\ \textit{deduce}}{\Gamma_2 := (\Gamma_1, a, b : \texttt{Integer} \wedge x + y = 2 * a + 2 * b) \vdash even(x + y)}\ 5.\ \text{deduce}}{\Gamma_1 := (\Gamma_0, (x, y : \texttt{Integer} \wedge even(x) \wedge even(y))) \vdash even(x + y)}\ 4.\ \textit{let, assume(Even\_Number)}}{\Gamma_0 \vdash \forall x, y : \texttt{Integer}(even(x) \wedge even(y) \Rightarrow even(x + y))}\ 3.\ \textit{let,assume,show}$$

Figure 2: Semantics of the proof in figure 1 as a proof tree. The initial context Γ_0, which contains the useful definitions (including definition 1 and 2) and axioms, needed to validate this proof. Arguments to rules are as η in assume(η) are justification(s).

tree shown in figure 2.

Formally, a MathAbs' document is a non-empty sequence of definitions, axioms and theorems with their proofs. We use BNF notation to describe the syntax of MathAbs.

⟨*Document*⟩ ::= ⟨*Math*⟩ ; ⟨*Math*⟩ ; . . .
⟨*Math*⟩ ::= Axiom ⟨*Formula*⟩
| Definition ⟨*Definition*⟩
| Theorem ⟨*Theorem*⟩ ; Proof ⟨*Proof*⟩

An Axiom is simply a formula while the other constructs are described later in this section. The language of ⟨*Formula*⟩ is arbitrary (it is a parameter of MathAbs) and subject to change from one mathematical domain to another. In first-order logic or set theory, ⟨*Formula*⟩ is simply a language for first-order formulas. That means, ⟨*Formula*⟩ represents mathematical statements in various structural blocks (i.e. axiom, definition, theorem, etc). The mathematical statements are mainly atomic facts formed by predicates and functions represented by relations, properties and sometimes by operations in mathematical texts. For instance, in the texts from number theory we may have relations such as equality, inequality, etc; we may have properties such as positiveness, evenness, etc; and we may have operations such as dividing the equation by x, etc. In contrast, the language of axiom, definition and proof remains the same for all mathematical domains. See §3.1 for more details.

Consider the following axiom as an example, which is only a statement; while its MathAbs is simply a ⟨*Formula*⟩:

Axiom 1. 1 is a natural number.

Axiom. $1 \in$ Nat

The ⟨*Definition*⟩, ⟨*Assignment*⟩ and ⟨*Ident*⟩ are defined below.

⟨*Definition*⟩ ::= ⟨*Assignment*⟩ ⟨*Definition*⟩
| define ⟨*Formula**⟩ as ⟨*Formula*⟩

⟨*Assignment*⟩ ::= let ⟨*Ident*⟩:⟨*Type*⟩
| assume ⟨*Formula*⟩

⟨*Ident*⟩ ::= ⟨*String*⟩

Only function names with varying number of variables as parameter (e.g. f, $f(x)$, $f(x, y)$, . . .) are allowed in ⟨*Formula**⟩ as a left hand side of a definition. We will explain more when giving the semantics in §4.

The ⟨*Assignment*⟩ only deals with universal variables. In ⟨*Assignment*⟩, let allows to introduce a new variable in the sequent; while assume allows to add a new hypothesis to deal with conditional definitions (e.g. the definition of division requires the divider to be non zero). More details will appear later in this section, in the explanation of ⟨*Proof*⟩ language.

A ⟨*Type*⟩ appearing in ⟨*Assignment*⟩ is also a parameter of MathAbs. In set theory, a type would simply be a set and in first order logic it would be a predicate symbol. On the other hand, ⟨*Type*⟩ also have a characteristic coming from linguistic: a ⟨*Type*⟩ is a set or predicate whose elements have a generic common name. e.g. Integer, Rational, etc. We keep this notion of type not to loose the linguistic information. This notion does not exactly correspond to type theory and is not even very precise (for instance, linguistically it is not easy to decide whether *prime* is a type or just a property of numbers). It is one of the main problems if we would translate CLM to a typed framework such as Coq or Agda.

⟨*Type*⟩ ::= `Prop` | `Set` | `Integer`
| `Number` | `Rational` | `Irrational`
| `NoType` | `Nat` | ⟨*Ident*⟩

Two examples of definitions and their MathAbs translation could be seen in 1-2 lines of figure 1.

A theorem could have zero or more assignment statements, followed by a show statement:

⟨*Theorem*⟩ ::= ⟨*Assignment*⟩ ⟨*Theorem*⟩
| show ⟨*Formula*⟩

An example theorem and its MathAbs could be seen in line 3 of figure 1. In theorems, introduction of variables and assumptions using the same language as in proofs is commonly found in mathematical texts. Therefore in the definition of theorem shown above, ⟨*Assignment*⟩ is allowed to preserve its NL structure.

The language of proof is substantially richer than the language of theorem, definition and axiom as shown below:

⟨*Proof*⟩ ::= trivial ⟨*Hint*⟩
| show ⟨*Formula*⟩ ⟨*Hint*⟩ ⟨*Proof*⟩
| deduce ⟨*Formula*⟩ ⟨*Hint*⟩ ⟨*Proof*⟩
| ⟨*Assignment*⟩ ⟨*Hint*⟩ ⟨*Proof*⟩
| { ⟨*Proof*⟩; ⟨*Proof*⟩; . . . } ⟨*Hint*⟩ *(split rule)*
| unfinished
| • ⟨*Proof*⟩

Here are explanations of ⟨*Proof*⟩ language constructs:

- As mentioned earlier, let allows to introduce a new variable in the sequent. For instance, "let x be an integer" is represented as "let x : `Integer`".
- Similarly, assume allows to add a new hypothesis in the sequent. For instance, "let x be a positive even integer" is represented as "let x : `Integer` assume $positive(x) \wedge even(x)$".
- show allows to change the conclusion of the sequent. It should be understood as "it is enough to show" or "we must prove". For instance, the sentence "prove that x is a positive integer" is represented as "let x : `NoType` show x : `Integer` $\wedge\, positive(x)$", if x is not declared in the previous statements. Similarly, the sentence "show that there is an integer x such that $x > 0$" is represented as "show $\exists x(x$: `Integer` $\wedge\, x > 0)$.
- trivial shows that the proof of the active sequent is finished successfully. In mathematical texts it would be represented by statements such as "this end the proof", "it is trivial", etc. However, we use this keyword at the end of all branches of a proof.
- unfinished shows that the proof of the active sequent is not finished but the rest of proof will be provided by the user later.
- full-stop (•) marks the end of a sentence in mathematical text. This information could be useful specially when we translate MathAbs in first-order formulas. Again, this information is kept to make the MathAbs' proof closer to the original proof in NL.
- { ⟨*Proof*⟩; ⟨*Proof*⟩; . . . } (split) is used for case analysis. It allows to split a proof in cases to use *proof-by-exhaustion* method. By using split, the active sequent is replaced by two or more sequents. ⟨*Proof*⟩; ⟨*Proof*⟩; . . . denotes a list with at least two items which themselves are ⟨*Proof*⟩. Consider the following example, showing a proof by case having 3 cases and its MathAbs; complete example is given in appendix.

  Proof. If $n = 0$ then $m = 1$. we can choose $u = 0$ . . .
  Otherwise if $m = 0$ then $n = 1$ . . .
  Otherwise there exist r and q such that . . .

  Proof. {
  assume $n = 0$ show $m = 1$ trivial by form $u := 0 \ldots$;
  assume $n \neq 0$ assume $m = 0$ deduce $n = 1$ . . . ;
  assume $n \neq 0$ assume $m \neq 0$ deduce $\exists r, q(\ldots)$ } ;

- deduce allows to deduce something from the existing hypotheses. In concrete terms, deduce $\mathbb{A}$ . . . is a syntactic sugar for: { show $\mathbb{A}$ trivial ; assume $\mathbb{A}$ . . . }, where $\mathbb{A}$ is an arbitrary formula. It means that we distinguish two branches in the proof by replacing the active sequent by two sequents. In the first branch we prove a statement and then, in the second sequent, we use it as an hypothesis. For instance, ". . . we conclude that $x = 10$." is translated as ". . . deduce $x = 10$" which is a syntactic sugar for ". . . { . . . show $x = 10$ trivial ; assume $x = 10$ . . . }"

We have three important operations in MathAbs proof as shown below:

Branching. We divide a proof in sub proofs by using split, deduce and trivial, where trivial is zero-branch sub-proof.

Evolution. We update the context by using let and assume and update the goal by using show.

Justifications. ⟨*Hint*⟩ appeared in almost all constructs of MathAbs ⟨*Proof*⟩. In informal mathematical proofs, it is common to give some justification for each statement or proof step. In MathAbs, these justifications are also preserved and we call them ⟨*Hint*⟩. At the time of translation from MathAbs to the language of a certain proof assistant, it should be used as a supporting argument. However, a lot of work is required before we can use them in proof checking using an automated theorem prover (ATP) or a proof assistant.

⟨*Hint*⟩ ::= by form ⟨*Formula*⟩ ⟨*Hint*⟩
| by axiom ⟨*Ident*⟩ ⟨*Hint*⟩
| by def ⟨*Ident*⟩ ⟨*Hint*⟩
| by oper ⟨*Ident*⟩ ⟨*Hint*⟩
| ϵ *(empty)*
| . . .

It is quite liberal in the sense that it is open to new constructs as shown by dots (. . .). Following are examples explaining them:

1. A formula e.g. $even(a)$ in a sentence such as "since, a is even, a^2 is even" (. . . deduce $even(a^2)$ trivial by $even(a)$)
2. A definition or axiom e.g. "we conclude that a is even by the definition of even number" (. . . deduce $even(a)$ trivial by def `Even_Number`)
3. An operation e.g. *squaring both sides* that should be translated into a prover specific tactic or command.

3.1 MathAbs and Extensibility

As evident from the MathAbs' syntax and as also shortly described in the explanation of axiom in §3, the language of axiom, definition and proof remains the same for all mathematical domains. In contrast, the language of ⟨*Formula*⟩, which represents mathematical statements, is arbitrary and subject to change from one mathematical domain to another.

In other words, the language of axiom, definition and proof is universal in mathematics; only the language for statement is domain dependent. That is why the various proof methods such as "proof by induction", "proof by contradiction", "proof by case", etc are also common to all mathematical domains. Hence, the above discussion leads to the conclusion that the

language of axiom, definition and proof in MathAbs need not to be extensible; only the language of $\langle \mathit{Formula} \rangle$ should be extensible.

4. THE MATHABS SEMANTICS

The semantic of a MathAbs document D is a pair $(\mathbb{T}(D), \mathbb{P}(D))$. $\mathbb{T}(D)$ is the theory of the documents, that is all its axioms and definitions as set of formulas, it is used as an initial context in all proofs. $\mathbb{P}(D)$ are the theorems together with their proof-trees.

DEFINITION 1 (AXIOM). *Axioms are just added to the theory:*

$$\mathbb{T}(D; \mathit{Axiom}\ A) := \mathbb{T}(D) \cup \{A\} \qquad \mathbb{P}(D; \mathit{Axiom}\ A) := \mathbb{P}(D)$$

DEFINITION 2 (DEFINITION). $\mathbb{D}(\theta)$ *takes a definition* θ *to build a MathAbs formula expressing the definition as an equivalence.*

We define its semantics as follows:

1. $\mathbb{D}(\mathsf{let}\ x{:}T\ \theta) := \forall x{:}T\ \mathbb{D}(\theta)$

   This definition requires that the language of formulas allows universal quantification.

2. $\mathbb{D}(\mathsf{assume}\ H\ \theta) := (H \Rightarrow \mathbb{D}(\theta))$

   This definition requires that the language of formulas contains implication.

3. $\mathbb{D}(\mathsf{define}\ A\ \mathsf{as}\ B) := (A = B)$

   where A and B are arbitrary formulas or terms. Recall that A should follow some restrictions for this to be a definition as mentioned in §3. In contrast, having no restrictions at all could not be troublesome either; if the definitions are understood as arbitrary equational axioms, possibly with some conditions which allows to handle partial definition.

   Remark: If A and B are formulas, the language should support equality on formulas or one should replace equality by equivalence in this case and distinguish two kind of definitions. We prefer to consider that the language of formulas allows equality.

   We can now give the semantics of definitions which is simply to add $\mathbb{D}(\theta)$ in the theory:

$$\begin{array}{lcl} \mathbb{T}(D; \mathsf{Definition}\ \theta) & := & \mathbb{T}(D) \cup \{\mathbb{D}(\theta)\} \\ \mathbb{P}(D; \mathsf{Definition}\ \theta) & := & \mathbb{P}(D) \end{array}$$

DEFINITION 3 (Proof). $\mathbb{S}(\pi;\ \Gamma \vdash G)$ *takes a proof* π *and an initial sequent* $(\Gamma \vdash G)$*, and build a proof tree whose conclusion is the initial sequent.*

We use the letter Γ for *context* that contains a list of variables with types and formulas, letters H, G for formulas and η is for $\langle \mathit{Hint} \rangle$. Here is the semantics for MathAbs proof:

1. $\pi = \mathsf{let}\ x : T\ \ \eta\ \ \pi'$,

   $$\mathbb{S}(\pi;\ \Gamma \vdash G) := \frac{\mathbb{S}(\pi';\ \Gamma, x : T \vdash G)}{\Gamma \vdash G}\ \mathit{let}(\eta)\ ,$$

   Here, $(\Gamma, x : T)$ means that variable x with its type T is added in the *context* Γ as an hypothesis and π' is the rest of the proof.

2. $\pi = \mathsf{assume}\ H\ \ \eta\ \ \pi'$,

   $$\mathbb{S}(\pi;\ \Gamma \vdash G) := \frac{\mathbb{S}(\pi';\ \Gamma, H \vdash G)}{\Gamma \vdash G}\ \mathit{assume}(\eta)$$

   Here an arbitrary formula H is added in the *context* as an hypothesis.

3. $\pi = \mathsf{show}\ G_2\ \ \eta\ \ \pi'$,

   $$\mathbb{S}(\pi;\ \Gamma \vdash G_1) := \frac{\mathbb{S}(\pi';\ \Gamma \vdash G_2)}{\Gamma \vdash G_1}\ \mathit{show}(\eta)$$

   Where G_1 is the previous goal which is changed to the new goal G_2.

4. $\pi = \mathsf{trivial}\ \eta, \qquad \mathbb{S}(\pi;\ \Gamma \vdash G) := \dfrac{}{\Gamma \vdash G}\ \mathit{trivial}(\eta)$

5. $\pi = \mathsf{unfinished}, \qquad \mathbb{S}(\pi;\ \Gamma \vdash G) := \dfrac{\vdots}{\Gamma \vdash G}\ \mathit{unfinished}$

   The vertical dots are here to denote the missing part of the proof-tree.

6. $\pi = \bullet\ \ \pi', \qquad \mathbb{S}(\pi;\ \Gamma \vdash G) := \mathbb{S}(\pi';\ \Gamma \vdash G)$

   i.e. full-stops are ignored. Again, this information could be useful specially when we translate MathAbs in first-order formulas.

7. $\pi = \{\pi_1, \ldots, \pi_n\}\ \eta$ *(i.e.* split*)*

   For all $i \in \{1, \ldots, n\}$, we find a proof tree $\mathbb{T}_i$ such that:

   $$\mathbb{S}(\pi_i;\ \Gamma \vdash G) = \frac{\mathbb{T}_i}{\Gamma \vdash G}\ r_i \text{ then}$$

   $$\mathbb{S}(\pi_1, \ldots, \pi_n;\ \Gamma \vdash G) := \frac{\mathbb{T}_1\ \ \ldots\ \ \mathbb{T}_n}{\Gamma \vdash G}\ \mathit{split}(r_1, \ldots, r_n, \eta)$$

   We keep the information regarding rules applied in proof at this step in $\mathsf{split}(r_1, \ldots, r_n, \eta)$.

   Remark: For all proof π and sequent $(\Gamma \vdash G)$ there is a unique proof tree $\mathbb{T}$ and rule r such that:

   $$S(\pi;\ \Gamma \vdash G) = \frac{\mathbb{T}}{\Gamma \vdash G}\ \mathrm{r}$$

DEFINITION 4 (THEOREM). $\mathbb{D}'(\sigma)$ *takes a theorem* σ *and build its statement. This definition is almost the same as* $\mathbb{D}(\theta)$ *in definition 2, hence the similar choice for the name.*

1. $\mathbb{D}'(\mathsf{let}\ x{:}T\ \sigma) := \forall x{:}T\ \mathbb{D}'(\sigma)$
2. $\mathbb{D}'(\mathsf{assume}\ H\ \sigma) := (H \Rightarrow \mathbb{D}'(\sigma))$
3. $\mathbb{D}'(\mathsf{show}\ A) := A$

DEFINITION 5 (THEOREM WITH PROOF). *Now, we can give the interpretation of a theorem* (σ) *with its proof* (π) *in a document* (D)*:*

$$\begin{array}{lcl} \mathbb{T}(D; \mathit{Theorem}\ \sigma; \mathit{Proof}\ \pi) & := & \mathbb{T}(D) \\ \mathbb{P}(D; \mathit{Theorem}\ \sigma; \mathit{Proof}\ \pi) & := & \mathbb{P}(D) \cup \mathbb{S}(\sigma\pi, \mathbb{T}(D) \vdash \mathbb{D}'(\sigma)) \end{array}$$

In this definition, the theory is not changed (as mentioned in the first line). For theorem with proof (in second line), we first compute $\mathbb{D}'(\sigma)$*, which is the statement of the theorem; then we reuse the definition* σ *at the beginning of the proof.*

5. COMPLETENESS

Various complete notions of proofs have been proposed such as natural deduction, sequent calculus, Hilbert systems, etc. Mathematicians use their rules in NL proofs implicitly; even without knowing sometimes. Here, we give a translation of natural deduction in MathAbs to establish its completeness. Moreover, we will see that some rules of natural deduction are decomposed in more than one rule by the semantics of MathAbs. Such rules are always logically correct.

As far as we know, some of the rules we present here[7] are rarely considered before.

Such decomposition of rules only occur because NL proofs may contain the intermediate steps. A sentence such as "we assume A and show B" is a typical example causing such intermediate steps as shown in the $\Rightarrow$-introduction rule below.

Such steps are harmless but can lead to problems easily. For instance instead of the above sentence, if we consider its logically equivalent sentence: "we show B and assume A", it will lead to an incorrect rule of MathAbs as it inverts assume and show. It gives us an insight that: "logically equivalent statements may not always be mathematically equivalent". It also demonstrates the fact that MathAbs tries to faithfully represent the NL proofs; as the later sentence will also not be easily accepted by mathematicians. So MathAbs and mathematical texts agree on this example.

When we split a sentence such as "we assume A and then we must show B" in two parts, it gives two correct but bizarre rules; which is interesting from a linguistic perspective. It also means that MathAbs' interpretation of NL proofs seems to enlighten the linguistics of mathematical English a little bit.

1. **$\Rightarrow$-introduction**

   When the statement to be proved is of the form $A \Rightarrow B$, then after the step of assuming A the statement to be proved will be B. This means that $\Rightarrow$-introduction may be represented as "assume A show B".
   Here is the MathAbs semantics, with some unusual rules:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, A \vdash B}}{\Gamma, A \vdash A \Rightarrow B}\ \text{show}}{\Gamma \vdash A \Rightarrow B}\ \text{assume}$$

2. **$\Rightarrow$-elimination (modus ponens):**

   This rule is translated as a split rule with two show constructs as "{ show $A \Rightarrow B$; show A }". The semantics is exactly the intended rule:

$$\dfrac{\dfrac{\vdots}{\Gamma, A \vdash B} \qquad \dfrac{\vdots}{\Gamma \vdash A}}{\Gamma \vdash A \Rightarrow B}\ \text{split(show,show)}$$

3. **$\forall$-introduction:**

   When the statement that need to be proved is of the form $\forall_x P(x)$, then after the step 'let y' the statement to be proved will be $P(y)$. So we represent $\forall$-introduction as "let $y : T$ show $P(y)$".

   Of course the name of the variable in the let usually will be the same as the name of the variable under the $\forall$ quantifier. Here is the semantics of this proof:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, y : T \vdash P(y)}}{\Gamma, y : T \vdash \forall x : T\ P(x)}\ \text{show}}{\Gamma \vdash \forall x : T\ P(x)}\ \text{let}$$

   We see again that the usual rule of natural deduction is decomposed in two steps. It may look strange but these steps are logically valid when y is not free in the conclusion sequent of the proof.

4. **$\forall$-elimination:** It is similar to the $\Rightarrow$-elimination. If we extend justifications ($\langle Hint \rangle$) to hold the value of the variable, it may be represented as "show $\forall x\ P(x)$ by replace(x, t)".

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash \forall x\, P(x)}}{\Gamma \vdash P(t)}\ \text{show(replace(x,t))}$$

5. **$\exists$-introduction**

   Its translation is similar to $\forall$-elimination: "show $P(t)$ by replace(x, t)". We expect that t is given immediately after $\exists x\ P(x)$.

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash P(t)}}{\Gamma \vdash \exists x\ P(x)}\ \text{show(replace(x,t))}$$

   However, sometimes NL proofs in mathematical texts do not give t immediately after $\exists x\ P(x)$; especially when it results from a long computation. We cannot handle such case in the current version of MathAbs (cf conclusion).

6. **$\exists$-left**

   Instead of $\exists$-elimination rule of natural deduction, mathematicians implicitly tend to use the left rule of sequent calculus in NL proofs. It is so because the left rule of sequent calculus is more natural.

   $\exists$ introduction at left (as an hypothesis) is represented as "let y : T assume $P(y)$"

$$\dfrac{\dfrac{\Gamma, \exists x : T\ P(x), y : T, P(y) \vdash \Delta}{\Gamma, \exists x : T\ P(x), y : T \vdash \Delta}\ \text{assume}}{\Gamma, \exists x : T\ P(x) \vdash \Delta}\ \text{let}$$

   Again, the rule is decomposed, but correct if y is not free in the conclusion sequent.

7. The reader can easily complete this with the rules for other connectives and even consider all sequent calculus rules.

6. CONCLUSION AND FURTHER WORK

In this paper, we gave a detailed account of MathAbs, its formal definition and semantics. The usefulness of an intermediary language between the natural language of the mathematician and the formal language of the logician is evident. MathAbs is a formal language which does not contain natural language elements at all, yet tries to faithfully represent the language of mathematics. Furthermore, the view of a mathematical proof as a way of explaining how the "context" evolves seems promising.

In terms of processing, it is intended to be simpler than the language of mathematics. One does not have to learn MathAbs, as it is a part MathNat system and is not visible to the end-user.

To answer the question of MathAbs' abstraction being expressive enough, the work presented here seems adequate

[7] Rules in the decomposition of $\Rightarrow$-introduction and $\forall$-introduction.

in principle. However in practice, we have formalized only a few examples in MathAbs, which may not be sufficient to make a definite conclusion. At least we can safely claim that the initial results seems very promising.

In future, the following three directions could be considered. First, from the semantics above, for each rule we can construct its equivalent formula and send it to the automated theorem prover (ATP) for validation. If all proof-steps are validated then the proof is valid and therefore establishes the truth of the theorem.

However, the proof search will often fail because of the unnecessary hypotheses and definitions that can dramatically extend the search space of ATP. Therefore, a future direction would be to design a language for justifications ($\langle Hint \rangle$) which not only reflects the information of the NL proof but can also be used by ATP. Such work will allow us to check more complex proofs.

Second, guiding ATP with justifications for equational reasoning is even more difficult and requires further investigation.

Finally, the introduction of meta-variables[8] in mathematics to take into account the use of known and unknown variables would be worthwhile and interesting. However, such work will not be easy because it prohibits to check each rule in a proof tree separately; as the check becomes global.

7. REFERENCES

[1] H. Barendregt. Towards an interactive mathematical proof language. In F. D. Kamareddine, editor, *Thirty-five Years of Automating Mathematics*, volume 28 of *Applied Logic series*, pages 25–36. Kluwer Academic Publishers, 2003.

[2] N. G. de Bruijn. The mathematical vernacular, a language for mathematics with typed sets. In *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*, pages 865–935, 1994.

[3] G. Frege. *Bergriffsschrift: eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Halle, 1879.

[4] K. Gödel. Die vollständigkeit der axiome des logischen funktionenkalküls (in german). *Monatshefte für Mathematik*, 37:349–360, 1930.

[5] G. Hardy and E. Wright. *An Introduction to the Theory of Numbers*. Oxford University Press, 4th edition, 1975.

[6] M. Humayoun and C. Raffalli. Mathnat - mathematical text in a controlled natural language. In A. Gelbukh, editor, *Special issue: Natural Language Processing and its Applications, Journal on Research in Computing Science*, volume 46, pages 293–310. CICLing-2010, 2010. ISSN 1870-4069.

[7] F. Kamareddine and J. B. Wells. Computerizing mathematical text with mathlang. *Electronic Notes in Theoretical Computer Science*, 205:5–30, 2008.

[8] M. Maarek. *Mathematical documents faithfully computerised: the grammatical and text and symbol aspects of the MathLang framework*. PhD thesis, Heriot-Watt University, June 2007.

[9] C. Raffalli. The phox proof assistant. `http://www.lama.univ-savoie.fr/~RAFFALLI/af2.html`, 2005.

[10] P. Thévenon. Validation of proofs using phox. *Electronic Notes in Theoretical Computer Science*, 140:55 – 66, 2005. Proceedings of the Second Workshop on Computational Logic and Applications (CLA 2004).

[11] P. Thévenon. *Vers un assistant à la preuve en langue naturelle*. PhD thesis, Université de Savoie, France, 2006.

[12] M. Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In *12th International Conference on Theorem Proving in Higher Order Logics*, volume 1690 of *Lecture Notes in Computer Science*, pages 167–184. Springer, 1999.

[13] F. Wiedijk. Formal proof sketches. In *TYPES 2003*, volume 3085 of *Lecture Notes in Computer Science*, pages 378–393. Springer, 2003.

[8] Alternatively we can call them existential variables.

APPENDIX

Example 1. Proof by case partially described in §3.

Theorem. Assume that m and n are relatively prime integers. Suppose that either $m \neq 0$ or $n \neq 0$. Then prove that there exist two integers u and v such that $u * n + v * m = 1$ holds.
Proof. *If* $n = 0$ then $m = 1$ because m and n are coprime. We can choose $u := 0$ and $v := 1$.
Otherwise if $m = 0$ and $n = 1$ then we can choose $u := 1$ and $v := 0$.
Otherwise there exist r and q such that $n = m * q + r$ holds by euclidean division. It is obvious that m and r are coprime and $r < m$. So by induction hypothesis there are u' and v' such that $u' * m + v' * r = 1$ holds. It implies that $u' * m + v' * (n - m * q) = v' * n + (u' - v' * q) * m = 1$. So we can choose $u := v'$ and $v := u' - v' * q$.

MathAbs.

Theorem. let m, n : Integer assume $coprime(m, n)\bullet$
assume $m \neq 0$ assume $n \neq 0\bullet$show $\exists(u, v : \text{Integer})(u*n+v*m = 1)\bullet$
Proof.
{
assume $n = 0$ deduce $coprime(m, n)$ show $m = 1$ by form $coprime(m, n)$ trivial by form $u := 0$ by form $v := 1\bullet$;

assume $n \neq 0$ assume $m = 0$ assume $n = 1$ trivial by form $u := 1$ by form $v := 0\bullet$;

assume $n \neq 0$ assume $m \neq 0$ assume $n \neq 1$ deduce $\exists(r, q : \text{NoType})(n = m * q + r)$ by def Euclidean_Division$\bullet$ deduce $coprime(m, r) \wedge r < m$ deduce $\exists(u', v' : \text{NoType})(u' * m + v' * r = 1)$ by def Induction_Hypothesis$\bullet$ deduce $u' * m + v' * (n - m * q) = v' * n + (u' - v' * q) * m = 1$ trivial by form $u := v'$ by form $v := u' - v' * q\bullet$
} ;

Example 2. Hardy & Wright's Theorem 3 [5, p. 21].

1. **Theorem.** If p is prime and $p \mid a * b$, then $p \mid a$ or $p \mid b$.
2. **Proof.** Suppose that p is prime and $p \mid a * b$.
3. If $p \nmid a$ then $gcd(a, p) = 1$; and therefore, by theorem 24, there are x and y such that $x * a + y * p = 1$ or† $x * a * b + y * p * b = b$ hold.
4. We conclude that $p \mid b$ because $p \mid a * b$ and $p \mid p * b$.

MathAbs.

1. **Theorem.** let a, b : NoType let p : Prime assume $p \mid a * b$ show $p \mid a$ or $p \mid b$ $\bullet$
2. **Proof.** let p : Prime assume $p \mid a * b$ $\bullet$
3. assume $p \nmid a$ show $gcd(a, p) = 1$ deduce $\exists(x, y : \text{NoType})((x*a+y*p = 1) \vee (x*a*b+y*p*b = b))$ by thm theorem_24 $\bullet$
4. deduce $p \mid a * b$ deduce $p \mid p * b$ deduce $p \mid b$ by form $p \mid a * b$ by form $p \mid p * b$ trivial $\bullet$

Note: the "or" marked with † in sentence 3 is a little bit problematic. Here it means "which implies", but in MathAbs it is treated as logical disjunction ($\vee$). However, the MathAbs proof is still mathematically correct. It is so because in the case when "$A \Rightarrow B$" is true, "$A \vee B$ becomes equivalent to B. It might the reason why Hardy & Wright used "or" instead of "which implies" in this proof.