MathNat - Mathematical Text in a Controlled Natural Language

Muhammad Humayoun and Christophe Raffalli

Laboratory of Mathematics (LAMA)
Université de Savoie, France
{mhuma, raffalli}@univ-savoie.fr*

Abstract. The MathNat[1] project aims at being a first step towards automatic formalisation and verification of textbook mathematical text. First, we develop a controlled language for mathematics (CLM) which is a precisely defined subset of English with restricted grammar and dictionary. To make CLM natural and expressive, we support some complex 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.
Second, we automatically translate CLM into a system independent formal language (MathAbs), with a hope to make MathNat accessible to any proof checking system. Currently, we translate MathAbs into equivalent first order formulas for verification.
In this paper, we give an overview of MathNat, describe the linguistic features of CLM, demonstrate its expressive power and validate our work with a few examples.

Key words: Mathematical Discourse, Informal Proofs, Anaphora, Controlled Language, Formalisation

1 Introduction

Since Euclid, mathematics is written in a specific scientific language which uses a fragment of a natural language (NL) along with symbolic expressions and notations. This language is structured and semantically well-understood by mathematicians but still not precise enough for automatic formalisation. By "not precise enough", we mean:

– Like any natural language text, mathematical text contains complex linguistic features such as anaphoric pronouns and references, rephrasing of a sentence in multiple ways producing canonical forms, proper handling of distributive and collective readings, etc.

* This work is funded by "Informatique, Signal, Logiciel Embarqué" (ISLE), Rhone-Alpes, France. http://ksup-gu.grenet.fr/isle/

[1] http://www.lama.univ-savoie.fr/~humayoun/phd/mathnat.html

– To make text comprehensive and aesthetically elegant, mathematicians tend to omit obvious details. Such reasoning gaps may be quite easy for a human to figure out but definitely not trivial for a machine.

In the current state of art, mathematical texts are sometimes formalised in very precise and accurate systems using specific formalisms normally based on some particular calculus or logic. Such a formal piece of mathematics does not contain natural language elements at all. Instead, it contains a lot of technical details of the underlying formal system, making it not suitable for human comprehension. This wide gap between textbook and formal mathematics, reduces the usefulness of computer assisted theorem proving in learning, teaching and formalising mathematics.

In this paper we focus on the first difficulty by developing a controlled language with the look and feel of textbook mathematics. We name it CLM (Controlled Language of Mathematics). It is a precisely defined subset of English with a slightly restricted grammar and lexicon. To make it natural and expressive enough, we support the above mentioned linguistic features. Here are the three main components of MathNat:

1. **The Controlled language of Mathematics (CLM):** Translate the NL text into an abstract syntax tree (AST) in two steps:
   (a) **Sentence Level Grammar:** It is sentence level (without context), attributed grammar with dependent records which we implement in Grammatical Framework (GF) [11]. We describe it in section 3 and 4.
   (b) **Context building:** We build context from the CLM discourse and solve the above mentioned linguistic features. Context building is described in section 5 and the linguistic features are described in section 6.
2. **Translation to MathAbs:** We automatically translate the AST into a system independent formal language (MathAbs), with a hope to make MathNat accessible to any proof checking system, described in section 7.
3. **Proof checking:** Currently, we translate MathAbs into equivalent first order formulas for automatic verification by automated theorem provers (ATP), described in section 7. This step is problematic since most ATP cannot verify even very simple proofs without help from the user. Further, (1) ATP are very sensitive to such hypotheses or details whose sole purpose is to offer an explanation to the reader. (2) Proofs in NL never give the exact list of hypotheses and definitions necessary at each step. This paper does not cover these problems.

The overall picture of the MathNat project is shown in figure 1.

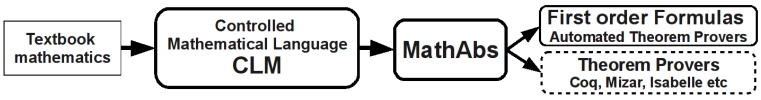

Fig. 1. MathNat - Overall picture

Such a controlled language is definitely easier to read than a formal language used by proof assistants. But is it easier to write? A realistic answer is negative

because a writer may go out of the scope of our grammar very quickly. However, an ambitious answer could be positive. Because the design of CLM supports incremental extendability of the grammar. Further, appropriate tools such as word completion, etc could help the writer to remain in the scope of the grammar.

But even if it fails to give enough freedom to an author for writing mathematics using CLM, we can still consider this work as a first step towards an almost complete mathematical language parser. Although, further work will be needed to extend the coverage tremendously, resolve more linguistic features and solve complexity issues that will certainly arise. With this in mind, the reader should therefore consider this article as a "proof of concept".

2 The Language of textbook mathematics

Mathematical texts mainly consists of theorems, lemmas and their proofs along with some supporting axioms, definitions and remarks. Axioms and definitions normally consist of a few statements expressing a proposition or a definitional equality. On the other hand, a proof is a collection of arguments presented to establish the truth of a proposition. It mainly follows a narrative style and its structure mostly remains the same for all mathematical domains.

The text in figure 2 is the main example that we'll use in this paper to illustrate the possibilities of MathNat. Sentences are numbered for future references. Here are a few remarks, general for mathematical text, showing that this text is already quite hard to formalise automatically: it mixes NL with symbolic expressions; it uses anaphoric pronouns. e.g. at line 7, 12; the use of explicit, implicit references or both e.g. at line 9, 10 and 6 respectively; a lot of keywords are used in the text (e.g. *let, suppose that, then, thus,* etc) that are mostly part of specific patterns such as:"if *proposition* then *proposition*", "(let | suppose | Thus | ...) *proposition* (because | by | ...) *proposition*"; the use of subordinates. e.g. at line 8, noun adjuncts. e.g. "*with no common factor*" at line 4 and explicit quantification. e.g. "for every x, if x is even then $x+1$ is odd" or "there is an integer x such that $x > 0$"; ...

1. **Definition 1.** x is a rational number if it is expressed as $\frac{p}{q}$, where p and q are integers with $q > 0$. [...]
2. **Theorem 1.** *Prove that* $\sqrt{2}$ *is irrational.*
3. **Proof.** Assume that $\sqrt{2}$ is a rational number.
4. By the definition of rational numbers, we can assume that $\sqrt{2} = \frac{a}{b}$ where a and b are non-zero integers with no common factor.
5. Thus, $b\sqrt{2} = a$.
6. Squaring both sides yields $2b^2 = a^2$ (1).
7. a^2 is even because it is a multiple of 2.
8. So we can write $a = 2c$, where c is an integer.
9. We get $2b^2 = (2c)^2 = 4c^2$ by substituting the value of a into equation 1.
10. Dividing both sides by 2, yields $b^2 = 2c^2$.
11. Thus b is even because 2 is a factor of b^2.
12. If a and b are even then they have a common factor.
13. It is a contradiction.
14. Therefore, we conclude that $\sqrt{2}$ is an irrational number.
15. This concludes the proof. □

Fig. 2. A typical math text

3 Sentence level CLM Grammar

GF [11] is a programming language for defining NL grammars that is based on Martin-Löf's dependent type theory [9]. We refer to [5] for further details. In GF,

we completely ignore the context, and design the CLM as an attributed grammar with dependent records. A GF grammar has two components: *abstract syntax* and *concrete syntax*. *Abstract syntax* defines semantic conditions to form abstract syntax trees (AST) of a language with grammatical functions (*fun*) making nodes of categories (*cat*). While a *concrete syntax* is a set of linguistic objects (strings, inflection tables, records) associated to ASTs, providing rendering and parsing. The process of translating an AST into one of its linguistic objects is called *linearization*.

```
cat Subject; Attribute; Prop;
fun MkProp: Subject -> Attribute -> Prop;
fun MkPronSubj: Pron -> Subject;
fun It: Pron; They: Pron;
```

Fig. 3. abstract syntax

Consider a part of our grammar for propositions such as "they are even integers". In figure 3, we define three categories. The function `MkProp` takes two parameters (a subject and an attribute) and forms a proposition. A subject is formed by pronouns `It` or `They`.

```
cat Property ; Type ;
fun MkAttrb:[Property] -> Type -> Attribute;
fun Even    : Property ;
fun Integer : Type ;
```

Fig. 4. abstract syntax

As shown in figure 4, `MkAttrb` function forms an attribute with a list of `Property` and `Type`. Next, we define functions `Even` and `Integer` of category `Property` and `Type` respectively. In full CLM grammar, we add properties (e.g. positive, odd, distinct, equal, etc) and types (e.g. rational, natural number, set, proposition, etc) in a similar fashion. To map this abstract syntax into its concrete syntax, we define a set of linguistic objects corresponding to the above categories and functions.

```
lincat Property   = {s : Str};
lin    Even       = {s ="even"};
param  Number = Sg | Pl ;
lincat Type   = {s : Number => Str};
lin Integer = {s = table{
                      Sg => "integer";
                      Pl => "integers"}};
lincat Pron = {s : Str ; n : Number} ;
lin    It   = {s = "it"   ; n = Sg};
lin    They = {s = "they" ; n = Pl} ;
```

Fig. 5. concrete syntax

In figure 5, the first line defines the linearization of `Property` category which is simply a string record. The second line shows this fact for the linearization of its function `Even`. The linearization of category `Type` is an inflection table (a finite function) from number to string, having one string value for each (singular and plural) as shown in fourth line. Its function `Integer` fills this table with two appropriate values in the next three lines.

```
lincat Attrb = {s : Number => Str};
lin MkAttrb props type = {s = table {
      Sg => artIndef ++ props.s ++ type.s!Sg;
      Pl =>             props.s ++ type.s!Pl}};
```

Fig. 6. concrete syntax

The linearization of pronoun `Pron` is a record, having a string and number. Further, we define the linearization of its functions and mention the fact that `It` is singular and `They` is plural, which will help us to make number agreement. Similar to category `Type`, `Attribute` is also an inflection table. Therefore, in figure 6, we define the linearization of its function `MkAttrb` accordingly. For instance, for singular value, we select the string value

of the category list of `Property` (props) with `(.s)`. Then, we select the singular string value of category `Type` with `(type.s!Sg)`. `(++)` concatenates these two strings with a space between them. `artIndef` makes an agreement for an indefinite article with the first letter of next word. e.g. producing *"an even number"* or *"a positive number"*. It is defined in GF resource library [12] which provides basic grammar for fourteen languages as an API.

Similarly, in figure 7, to form a proposition, in `MkProp`, we select appropriate string values of tables `be` and attribute by an agreement of number with subject, and concatenate them with subject. For instance, if we parse the propositions such as "they are even integers" and "it is an even integer", we'll get following **abstract syntax trees**:

```
lincat Prop = {s : Str};
oper be = {s = table{Sg => "is" ; Pl => "are"}};
lin MkProp subj attrb =
  {s = subj.s ++ be.s!subj.n ++ attrb.s!subj.n};
```

Fig. 7. concrete syntax

```
1. MkProp (MkPronSubj They) (MkAttrb (BaseProperty Even) Integer)
2. MkProp (MkPronSubj It) (MkAttrb (BaseProperty Even) Integer)
```

Fig. 8. abstract syntax trees

4 Synopsis of CLM Grammar

As a whole, CLM text is a collection of axioms, definitions, propositions (theorems, lemma, ...) and proofs structured by specific keywords (Axiom, Definition, Theorem and Proof). The text following these keywords are list of sentences obeying different GF grammars (one for axioms, one for definition, ...). These grammars are not independent and share a lot of common rules.

We present in this section a short but incomplete synopsis of the grammar for sentences allowed in theorems and proofs. We describe it in an abstract way with some examples and it obeys the following conventions: [text] means that *text* is optional, (text1|text2) means that both *text1* and *text2* are possible, dots (...) means that only few constructions are given due to space limitation, and each dash (–) represents a pattern in the grammar. We start this synopsis by extending the grammar of our running example:

1. `Exp` is a symbolic expression (equation not allowed).
   It is encapsulated by quotes to distinguish it from natural language parts of grammar. Defining a formal grammar for symbolic expressions and equations in GF is definitely possible but as it is specially designed for NL grammars, in past, it has caused serious efficiency overhead for parsing, because of CLM's size. Therefore we define a *Labelled BNF grammar* in bnfc tool [4]. Examples of `Exp` are $\sqrt{2}$ in line 2-3, x in line 1 and a^2 and 2 in line 7 of figure 2.
2. `Exps` is a list of `Exp`. e.g. a, b in line 12 of figure 2, etc.
3. `Subject`, as partially described before, is (`Exps` | anaphoric pronouns | ...)
4. `Attribute`, as partially described before, is (

`Quantity`, list of `Property` and `Type`. e.g. two positive even integers. three irrational numbers, etc | `Property` e.g. positive, even, distinct, equal, etc | `Quantity`, `Relation` and `Exp`. e.g. an element of y. two elements of z, etc |...)

5. `Prop` of *proposition* is (Positive and negative statements formed by `Subject` and `Attribute` e.g. at line 2, 3, etc | Statements formed by `Subject`, `Quantity` and `Relation`. e.g. x, y and z have a common factor, they have two common multiples, etc | Statements containing existential and universal quantification. | If then statements. | Disjunction statements. | ...)
6. `Eq` is a symbolic equation, encapsulated by quotes. As described for `Exp`, it is also defined as a *Labelled BNF grammar*. e.g. in line 4, 5, 6, 8, 9, 10, etc
7. `EqOrProp` is (`Eq` | `Prop`)
8. `DefRef` is (`Property` and `Type`. e.g the definition of even numbers, etc | `Property` e.g. the definition of evenness, etc)
9. `Operation` is (`Relation` e.g factoring both sides, taking square at both sides, etc | `Relation` and `Exp` e.g. dividing both sides by x, multiplying the equation by 2, etc | ...)
10. `Justif` is (`Eq` | `Prop` | `DefRef` | `Operation`)

Sentence for theorems could be described as follows:

11. *"Prove statement" with an optional subordinate*
- [(show|prove) [that]] `Eq` [holds] [(where|when|if|with a condition that|...) `EqOrProp`]
- [(show|prove) that] `Prop` [(where|when|if|with a condition that|...) `EqOrProp`]

e.g. "Prove that $x + y = x$ holds if $y = 0$", "If x is even then $x + 1$ is odd with a condition that $x > 0$", line 2 of figure 2, etc.
The main difference between these two patterns is "holds" which is optional for statements containing an equation, but does not appear in statements containing propositions. This applies to all CLM statements. So, in the following statements we mention these two patterns as one using `EqOrProp`

12. *Assumption with an optional subordinate*
- (let |[we] suppose [that]|we can write [that]|...) `EqOrProp` [holds] [(where|...) `EqOrProp`] e.g. "we suppose that $x + y > 0$", line 3 of figure 2, etc

Note: with the example of section 3, we can infer a statement such as "let x is an even number", which is grammatically incorrect. In fact, the actual grammar defined for propositions is a bit more complicated than what is given in section 3. In the actual grammar, attribute and proposition are inflection tables with two values; one for *let* statements ("be an even..."), and second for the remaining ("is/are ...").

13. *Sentences that cannot be the first sentence in a theorem and proof*
(then|thus|so|therefore), followed by any statement. e.g. line 5, 11 of figure 2, etc

Note: In theorem, statements of the form 12 and 13 are often stated to help the reader as a starting point for the proof.

Proof statements could be described as follows:

14. Shall-Prove statement with an optional subordinate
– we [(will | shall |have to)] prove [that] `EqOrProp` [holds] [(where|...) `EqOrProp`]
Shall-Prove pattern is almost the same as 11, but in proofs, with different keywords, we distinguish *goals* from *deductions*.

15. Assumption with an optional subordinate
Same as 12. e.g. line 8 of figure 2 (if we remove "So" from the sentence), etc

16. Assumption with a justification and an optional subordinate
– ([we] assume [that]|...) `EqOrProp` (since |because |by |...) `Justif` [(where |...) `EqOrProp`]
– (since |because |by |...) `Justif` ([we] assume [that]|...) `EqOrProp` [(where |...) `EqOrProp`] e.g. line 4 of figure 2, etc
These patterns are rough estimate of the actual coverage, it is possible to infer a statement which is grammatically incorrect. e.g. "assume $x + y = z$ because squaring both sides". In actual grammar, we have six patterns for 16 that ensures the grammatical correctness. In doing so, `Justif` is not just one category. Instead it is formed by some of these: (`Eq` |`Prop` |`DefRef` |`Operation`). This applies to all patterns of the grammar.

17. Deduction with an optional subordinate
– [(we (get |conclude |deduce |write |...) [that])] `EqOrProp` [holds] [(where|...) `EqOrProp`]
e.g. "we conclude that there is an even number x such that $x > 2$", line 5, 12 and 14 (if we remove "Therefore,") of figure 2, etc

18. Deduction with a justification and an optional subordinate
– (we get |...) `EqOrProp` (because |...) `Justif` [(where |...) `EqOrProp`]
– (because |...) `Justif` (we get |...) `EqOrProp` [(where |...) `EqOrProp`]
– `Operation` (yields | shows | establishes |...) `EqOrProp` [(where |...) `EqOrProp`]
– (because |...) `Justif`, `EqOrProp` [where `EqOrProp`]
e.g. line 6, 7, 9, 10, 11 (if we remove "Thus") of figure 2, etc

19. Proof by case (nested cases are allowed)

we proceed by case analysis	if *condition* then ...
case: *condition* ... [this ends the case.]	otherwise if *condition* then ...
case: *condition* ... [this ends the case.]	.....
..... [it was the last case.]	otherwise ...

5 Discourse Building

GF can compile the grammar into code usable in other general purpose programming languages. For instance, in Haskell, rendering functions *linearize* and

parse are made available as ordinary functions. It translates abstract syntax into algebraic data types forming objects of type AST.

When the math text is parsed by CLM parser, a list of sentence level AST is produced. We recognise each AST by pattern matching on algebraic data types and build the context from CLM discourse. For an AST, we record every occurrence of symbolic expressions, equations, pronouns and references as shown in Table 1. Further from Table 1, it seems like we keep a placeholder for each anaphoric pronoun that appears in the text. In fact, they are resolved immediately as they appear with the algorithm described in section 6. Context building and translation of math text into MathAbs are interleaved, as shown in the following procedure. So both are performed in the same step. However MathAbs translation is described in section 7. Context for theorem and proof in figure 2 is given in Table 1, 2 and 3.

Notations: S denotes an arbitrary AST (abstract syntax tree).
Context is 3-tuple `(CV, ST, CE)` where:
`CV` is a list of 4-tuple *(sentence number S_n, list of symbolic objects ($obj_1, ..., obj_n$), type, number)* as shown in table 1.
`ST` is a list of 3-tuple *(S_n, logical formula, type)* as shown in table 2.
`CE` is a list of 3-tuple *(S_n, equation, reference)* as shown in table 3.

Procedure: we start the analysis with an empty *Context* which evolves while examining the AST of each sentence. The following procedure is repeated until we reach the end of the text. Just a few cases are mentioned here due to the space limitations.

```
//an assumption or deduction containing an equation. e.g. line 4,8 or 5,6,9 respectively
If S matches (Assume Eq) or (Deduce Eq)
 mathabs_eq := translate Eq into MathAbs's formula
 left_eq := left side of mathabs_eq*
 lookup for a latest (S_n, obj, type, Sg) of CV if left_eq=obj
   t := if such (S_n, obj, type, Sg) found then type else NoType
 add (S_n, left_eq, t, Sg) in CV
 add (S_n, mathabs_eq, reference if provided in Eq) in CE
 add (S_n, mathabs_eq, st_type(S)) in ST
```

```
st_type(S) := if S matches (Assume ...) then Hypothesis //common to all
           else if S matches (Deduce ...) then Deduction else Goal
```

∗The choice of taking the left side of an equation is a bit naive. But there is no established algorithm to decide which identifier an anaphoric pronoun refers to. e.g. In "[...] we have $\forall_x(f(x) = p)$. Thus, it is a constant function" pronoun "it" refers to f, which is not at all trivial even for a human sometimes. So we always take the left side as a convention, which in fact makes sense for many equations.

```
//an assumption or deduction containing Exps e.g. line 2,3 or line 7,11 respectively
If S matches (Assume (MkProp ... Exps)) or (Deduce (MkProp ... Exps))
1. Statements** such as "we (assume|conclude) that x is a positive integer"
 type := given in S
 number := if (length Exps)=1 then Sg else Pl
```

```
add (S_n, Exps***, type, number) in CV
mathabs_exp := translate S into MathAbs's formula
add (S_n, mathabs_exp, st_type(S)) in ST
```

**We mention only one case due to space limitation.

***The convention of taking whole expression is also a bit naive. But again, there is no established algorithm to decide which identifier an anaphoric pronoun refers to. e.g. "[...] because $a|p$ (a divides p), it is a prime or 1" Here pronoun "it" refers to a. But in statement "[...] $2|b$. Thus it is even" pronoun it refers to b.

```
//a prove statement containing an equation. e.g. show that 2x + 2y = 2(x + y)
If S matches (Prove Eq)
mathabs_eq := translate Eq into MathAbs's formula
left_eq := left side of mathabs_eq
add (S_n, left_eq , NoType, Sg) in CV
add (S_n, mathabs_eq, reference if provided in Eq) in CE
vars := list of all variables appeared in mathabs_formula
not_decl_vars := all variables of vars that not found in MathAbs context
//e.g. 2x + 2y = 2(x + y) translated as ∀_{x,y}(2x + 2y = 2(x + y)) if x, y not found
mathabs_formula := ∀_{not_decl_vars}(mathabs_eq)
add (S_n, mathabs_formula, st_type) in ST
```

6 Linguistic features

Recall that `(CV, ST, CE)` is the *Context* defined in section 5. Also recall that in the course of context building, we resolve all kind of anaphora immediately as they appear with the following algorithm.

6.1 A Naive Algorithm for Anaphoric Pronouns

If S matches (...(...It) ...)
i.e. a statement containing pronoun "it". We replace this pronoun with the latest obj_1 of *(S_n, obj_1, Sg, type)* from `CV`. e.g. line 7 of figure 2 is interpreted as "a^2 is even because a^2 is a multiple of 2"

If S matches (...(...They) ...)

1. If no *quantity* (e.g. two, three, ...) is mentioned in S, we replace pronoun "they" with the latest $(obj_1, ..., obj_n)$ of *(S_n, ($obj_1, ..., obj_n$), Pl, type)* from `CV`. e.g. line 12 of figure 2 is interpreted as "If a and b are even then a and b have a common factor"
2. Otherwise, if there is a *quantity* Q mentioned in `S` then we replace this pronoun with the latest $(obj_1, ..., obj_n)$ of *(S_n, ($obj_1, ..., obj_n$), Pl, type)* when Q=*length*$(obj_1, ..., obj_n)$.
   e.g. the last statement from "suppose that $x + y + z > 0$.[...] assume that a and b are positive numbers.[...] they are three even integers" is interpreted as "[...] x, y and z are three even integers"

6.2 Anaphora of Demonstrative Pronouns

The <*type,number*> pair of *(S_n, (obj_1, ..., obj_n), number, type)* from `CV`, allows to solve anaphora for demonstrative pronouns "this" and "these". e.g. "*these* integers . . . ", is replace by the latest (obj_1, ..., obj_n) with *number=Pl* $\wedge$ *type=Integer*.

For the moment we only deal with the pronouns referring to expressions. Pronouns referring to propositions and equations are left as a future work.

Table 1. `CV`: Symbolic objects

S_n	Objects	Type	Number	S_n	Objects	Type	Number	S_n	Objects	Type	Number
2.	$\sqrt{2}$	`NoType`	`Sg`	7.	**It**	**?**	**Sg**	10.	b^2	`NoType`	`Sg`
3.	$\sqrt{2}$	Rational	Sg	7.	2	`NoType`	`Sg`	11.	b	`Integer`	`Sg`
4.	$\sqrt{2}$	`Rational`	`Sg`	8.	a	`Integer`	`Sg`	11.	2	`NoType`	`Sg`
4.	a, b	`Integer`	`Pl,2`	8.	c	`Integer`	`Sg`	11.	b^2	`NoType`	`Sg`
5.	$b\sqrt{2}$	`NoType`	`Sg`	9.	$2b^2$	`NoType`	`Sg`	12.	a, b	`Integer`	`Pl,2`
6.	$2b^2$	`NoType`	`Sg`	9.	a	`Integer`	`Sg`	12.	**They**	**?**	**Pl,?**
7.	a^2	`NoType`	`Sg`	10.	2	`NoType`	`Sg`	14.	$\sqrt{2}$	`Irrational`	`Sg`

Table 2. `ST`: Logical formulas

S_n	Logical formula	Stmnt type	S_n	Logical formula	Stmnt type
2.	$\sqrt{2} \notin Q$	`Goal`	7.	`even`(a^2)	`Deduction`
3.	$\sqrt{2} \in Q$	`Hypothesis`	8.	$c \in Z \wedge a = 2c$	`Hypothesis`
4.	$a, b \in Z \wedge$		9.	$2b^2 = (2c)^2 = 4c^2$	`Deduction`
	`positive`$(a) \wedge$		10.	$b^2 = 2c^2$	`Deduction`
	`positive`$(b) \wedge$	`Hypothesis`	11.	`even`(b)	`Deduction`
	`no_cmn_factor`(a, b)		12.	`even`$(a) \wedge$ `even`$(b) \Rightarrow$	`Deduction`
	$\wedge\sqrt{2} = a/b$			`one_cmn_factor`(a, b)	
5.	$b\sqrt{2} = a$	`Deduction`	13.	`False`	`Deduction`
6.	$2b^2 = a^2$	`Deduction`	14.	$\sqrt{2} \notin Q$	`Deduction`

Table 3. `CE`: Equations

S_n	Equation	Reference	S_n	Equation	Reference	S_n	Equation	Reference
4.	$\sqrt{2} = a/b$	`NoRef`	6.	$2b^2 = a^2$	`1`	9.	$2b^2 = (2c)^2 = 4c^2$	`NoRef`
5.	$b\sqrt{2} = a$	`NoRef`	8.	$a = 2c$	`NoRef`	10.	$b^2 = 2c^2$	`NoRef`

6.3 Solving References

1. Explicit reference to an equation as appeared at line 9. In the current settings of the context, it is trivial to solve such anaphora because each reference is preserved. e.g. line 9 is interpreted as "We get $2b^2 = (2c)^2 = 4c^2$ by substituting the value of a into $2b^2 = a^2$"
2. Implicit reference to an equation as appeared at line 6 and 10. At line 10, dividing both sides by 2 implies that there is an equation in some previous sentence. So we check this condition and if an equation is found in `CE`, we put it in place and interpret it as "Dividing $2b^2 = (2c)^2 = 4c^2$ by 2 at both sides, yields $b^2 = 2c^2$"

3. Reference to an equation that is mentioned far from the current sentence. e.g. "dividing the (last|first) equation by 2". It is also quite trivial to solve because we lookup `CE` for last or first element.
4. Reference to a hypothesis, deduction or statement. e.g. "we deduce that $x + y = 2(a + b)$ by the (last|first) (statement|hypothesis|deduction)".
   For instance, for a statement containing "by the last hypothesis", we simply pick the latest *formula* of *(S_n, formula, type)* from `ST` when *type=Hypothesis*. However, for a statement containing "by the last statement", we pick the latest *formula* of *(S_n, formula, type)* when *type=(Hypothesis or Deduction)*.

6.4 Distributive vs. Collective Readings

Like any natural language text, distributive and collective reading are common in math text and we deal with them in CLM appropriately. For example, the statements such as "x, y are positive" and "x, y are equal" are distributive and collective respectively. Some collective readings could be ambiguous. Consider the statement "a,b,c are distinct". Are variables a, b, c pair-wise distinct or just some of them distinct? Currently we neglect such ambiguity and always translate such properties as 2-arity predicates. Further, because collective readings require their subjects to be plural, statements such as "x is equal" are not allowed.

7 MathAbs and proof checking

The Haskell code supporting above mentioned linguistic features does more. It translates the AST given by GF to an abstract Mathematical Language (MathAbs). This language is a system independent formal language that we hope will make MathNat accessible to any proof checking system.

MathAbs (formally new_command) was designed as an intermediate language between natural language proofs and the proof assistant PhoX [10] in the DemoNat project [14][15]. Since new_command includes some of the standard commands of PhoX, we adapt it to MathNat by doing some minor changes.

MathAbs can represent theorems and their proofs along with supporting axioms and definitions. On a macro level, a MathAbs's document is a sequence of definitions, axioms and theorems with their proofs. However, analogous to informal mathematical text, the most significant part of MathAbs is the language of proof. While, the definitions and statements of propositions are a fraction of the text compared to the proofs.

A proof is described as a tree of logical (meta) rules. Intuitively, at each step of a proof there is an implicit active sequent, with some hypotheses and one conclusion, which is being proved and some other sequents to prove later. The text in NL explains how the active sequent is modified to progress in the proof and gives some justifications (hints). Similarly, a theorem forms the initial sequent with some hypotheses and one goal, which is then passed to the proof. While, axioms and definitions also form the initial sequent by adding them as hypotheses.

The basic commands of MathAbs are let to introduce a new variable in the sequent, assume to introduce a new hypothesis, show to change the conclusion of the sequent, trivial to end the proof of the active sequent and { ... ; ... } to branch in the proof and state that the active sequent should be replaced by two or more sequents. deduce A ... is a syntactic sugar for { show A trivial ; assume A ... }. We translate math text of figure 2 in MathAbs as shown below:

1. `Definition.` $r \in Q \Leftrightarrow \exists_{p,q \in Z}(r = \frac{p}{q} \wedge q > 0)$
2. `Theorem.` show $\sqrt{2} \notin Q$
3. `Proof.` assume $\sqrt{2} \in Q$
4. let $a, b \in$ `Z` assume $\sqrt{2} = a/b$ assume `positive(`a`)` $\wedge$ `positive(`b`)` $\wedge$ `no_cmn_factor(`a, b`) by def rational_Number`
5. deduce $b\sqrt{2} = a$
6. deduce $2b^2 = a^2$ `1 by oper squaring_both_sides(`$b\sqrt{2} = a$`)`
7. deduce `multiple_of(`a^2`, 2)`
   deduce `even(`a^2`) by form multiple_of(`a^2`, 2)`
8. let $c \in$ `Z` assume $a = 2c$
9. deduce $2b^2 = (2c)^2 = 4c^2$ `by oper substitution(`$a, 2b^2 = a^2$`)`
10. deduce $b^2 = 2c^2$ `by oper division(`$2, 2b^2 = (2c)^2 = 4c^2$`)`
11. deduce `factor_of (2,` b^2`)`
    deduce `even(`b`) by form factor_of(2,`b^2`)`
12. deduce `(even(`a`)` $\wedge$ `even(`b`))` $\Rightarrow$ `one_cmn_factor(`a`,` b`)`
13. show $\bot$ trivial

Remarks:

- At line 7, first, we deduce the justification i.e. `multiple_of(`a^2`, 2)`, and then deduce the whole statement. Same applies to 11.
- However, the above rule does not apply to definitional references and operations as shown in 4, 6, 9, 10.
- We can safely ignore Line 14 and 15 of figure 2 because the proof tree is already finished at line 13.

We can represent above MathAbs proof as a proof tree using arbitrary rules (not just the rules of natural deduction). Then, for each rule we can produce a formula that justifies it. The line 2-5 of above MathAbs can be read as the following proof tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_2 \vdash b\sqrt{2} = a} \quad \dfrac{\vdots}{\Gamma_3 \equiv (\Gamma_2, b\sqrt{2} = a) \vdash \sqrt{2} \notin Q}}{\Gamma_2 \equiv (\Gamma_1, a, b \in \texttt{Z},\ \sqrt{2} = a/b,\ \texttt{positive(}a\texttt{), positive(}b\texttt{), no\_cmn\_factor(}a, b\texttt{))} \vdash \sqrt{2} \notin Q}}{(\Gamma_1 \equiv \Gamma, \sqrt{2} \in Q) \vdash \sqrt{2} \notin Q}}{\Gamma \vdash \sqrt{2} \notin Q}$$

Fig. 9. MathAbs proof as a proof tree. Γ is a context which contains the useful definitions and axioms, needed to validate this proof.

As a first prototype we implemented a translation from MathAbs to first order formulas for validation. The above MathAbs is translated as follows (one formula for each rule):

3. $\vdash$ (rational($\sqrt{2}$) $\Rightarrow$ irrational($\sqrt{2}$)) $\Rightarrow$ irrational($\sqrt{2}$)
4. $\Gamma_1 \vdash \forall_{a,b}$((int($a$) $\wedge$ int(b) $\wedge$ $\sqrt{2} = a/b \wedge a, b > 0 \wedge$ gcd$(a, b) = 1) \Rightarrow irrational($\sqrt{2}$)) $\Rightarrow$ irrational($\sqrt{2}$)
   where $\Gamma_1 \equiv$ rational($\sqrt{2}$)
5. $\Gamma_2 \vdash (b\sqrt{2} = a \wedge (b\sqrt{2} = a \Rightarrow$ irrational($\sqrt{2}$))) $\Rightarrow$ irrational($\sqrt{2}$)*
   $\Gamma_2 \vdash b\sqrt{2} = a$
   where $\Gamma_2 \equiv \Gamma_1, \forall a, b$ (int(a)$\wedge$int(b) $\wedge$ $\sqrt{2} = a/b \wedge a, b > 0 \wedge$ gcd$(a, b) = 1$)
6. $\Gamma_3 \vdash (2b^2 = a^2 \wedge (2b^2 = a^2 \Rightarrow$ irrational($\sqrt{2}$))) $\Rightarrow$ irrational($\sqrt{2}$)*
   $\Gamma_3 \vdash 2b^2 = a^2$
   where $\Gamma_3 \equiv \Gamma_2, b\sqrt{2} = a$
7. $\Gamma_4 \vdash$ (multiple_of$(a^2, 2) \wedge$ (multiple_of$(a^2, 2) \Rightarrow$ irrational($\sqrt{2}$))) $\Rightarrow$ irrational($\sqrt{2}$)*
   $\Gamma_4 \vdash$multiple_of$(a^2, 2)$
   where $\Gamma_4 \equiv \Gamma_3, 2b^2 = a^2$
   $\Gamma_5 \vdash$ (even$(a^2) \wedge$ (even$(a^2) \Rightarrow$irrational($\sqrt{2}$))) $\Rightarrow$ irrational($\sqrt{2}$)*
   $\Gamma_5 \vdash$ even(a^2)
   where $\Gamma_5 \equiv \Gamma_4$, multiple_of$(a^2, 2)$
8. $\Gamma_6 \vdash \forall c$((int$(c) \wedge a = 2c) \Rightarrow$irrational($\sqrt{2}$)) $\Rightarrow$ irrational($\sqrt{2}$)
   where $\Gamma_6 \equiv \Gamma_5$, even(a^2)
9. $\Gamma_7 \vdash (2b^2 = (2c)^2 = 4c^2 \wedge (2b^2 = (2c)^2 = 4c^2 \Rightarrow$irrational($\sqrt{2}$))) $\Rightarrow$ irrational($\sqrt{2}$)*
   $\Gamma_7 \vdash 2b^2 = (2c)^2 = 4c^2$
   where $\Gamma_7 \equiv \Gamma_6$, int$(c) \wedge a = 2c$
10. $\Gamma_8 \vdash (b^2 = 2c^2 \wedge (b^2 = 2c^2 \Rightarrow$ irrational($\sqrt{2}$))) $\Rightarrow$ irrational($\sqrt{2}$)*
    $\Gamma_8 \vdash b^2 = 2c^2$
    where $\Gamma_8 \equiv \Gamma_7, (2b^2 = (2c)^2 = 4c^2)$
11. $\Gamma_9 \vdash$ (factor_of$(2, b^2) \wedge$(factor_of $(2, b^2) \Rightarrow$ irrational($\sqrt{2}$))) $\Rightarrow$ irrational($\sqrt{2}$)*
    $\Gamma_9 \vdash$ factor_of$(2, b^2)$
    where $\Gamma_9 \equiv \Gamma_8, (b^2 = 2c^2)$
    $\Gamma_{10} \vdash$ (even$(b)) \wedge$ (even$(b) \Rightarrow$ irrational($\sqrt{2}$))) $\Rightarrow$ irrational($\sqrt{2}$)*
    $\Gamma_{10} \vdash$ even(b)
    where $\Gamma_{10} \equiv \Gamma_9$, factor_of $(2, b^2)$
12. $\Gamma_{11} \vdash$ ((even$(a) \wedge$ even$(b) \Rightarrow$ one_cmn_factor$(a, b)) \wedge$ ((even$(a) \wedge$ even$(b) \Rightarrow$ one_cmn_factor$(a, b)) \Rightarrow$ irrational($\sqrt{2}$))) $\Rightarrow$ irrational($\sqrt{2}$)*
    $\Gamma_{11} \vdash$ even$(a) \wedge$ even$(b) \Rightarrow$ one_cmn_factor(a, b)
    where $\Gamma_{11} \equiv \Gamma_{10}$, even$(b)$
13. $\Gamma_{12} \vdash (\bot \Rightarrow$ irrational($\sqrt{2}$))
    $\Gamma_{12} \vdash \bot$
    where $\Gamma_{12} \equiv \Gamma_{11}$, (even$(a) \wedge$ even$(b) \Rightarrow$ one_cmn_factor$(a,$ $b))$

Remarks:

- Since deduce A is a syntactic sugar of { show A trivial ; assume A ... }, it produces a lot of tautologies of the form $(A \wedge (A \Rightarrow B)) \Rightarrow B$ in the first-order formulas. Where B is the main goal to prove. They are marked with * above.
- In proof, if we add in a sentence such as "proof by contradiction" this adds a MathAbs command show $\bot$ that would replace the conclusion irrational($\sqrt{2}$) by $\bot$.
- The justifications such as "by def rational_Number" that are preserved in MathAbs were removed from the first order translation, because most of the automated theorem provers are unable to use such justifications.

– In the above first order formulas, types are treated as predicates. The notion of types used in the grammar is linguistic and do not exactly correspond to the notion of type in a given type theory. This is one of the main problems if we want to translate CLM to a typed framework such as Coq[2] or Agda[3].

8 Related Work

AutoMath [2] of N.d. Bruijn, is one of the pioneering work in which a very restricted proof language was proposed. After that such restricted languages are presented by many systems. For instance, the language of Mizar, Isar [16] for Isabelle, the notion of *formal proof sketches*[17] for Mizar and Mathematical Proof Language *MPL* [1] for Coq. However such languages are quite restricted and non ambiguous having a programming language like syntax, with a few syntactic constructions. Therefore, like MathAbs, we consider them as an intermediate language between mathematical text and proof checking system.

The MathLang project [7] goes one step further by supporting the manual annotation of NL mathematical text. Once the annotation is done by the author, a number of transformations to the annotated text is automatically performed for automatic verification. So MathLang seems quite good in answering the second question raised in the introduction but neglects the possibility of NL parsing completely.

The work of Hallgren et al.[6] presents an extension to the type-theoretical syntax of logical framework Alfa, supporting a self extensible NL input and output. Like MathNat, its NL grammar is developed in GF but it does not support the rich linguistic features as we do. Similar to this work, we hope to make CLM grammar extensible in future.

Nthchecker of Simon [13] is perhaps the poineering work for its time, towards parsing and verifying informal proofs. However, according to Zinn [18] its linguistic and mathematical analysis is quite adhoc and we second his opinion.

In recent times, Vip - the prototype of Zinn [18] is a quite promising work. In his doctoral manuscript and papers, Zinn gives a good linguistic and logical analysis of textbook proofs. In Vip, he builds an extension of discourse representation theory (DRT) for parsing and integrates proof planning techniques for verification. Vip can process two proofs (nine sentences) from number theory. In our opinion, this coverage is too limited to verify the usefulness of presented concepts. Further, it supports limited linguistic features. Unfortunately, no further work has appeared after 2005. We do not use DRT because it is perhaps too complex and an overkill for the task of analysing mathematical text. However, we may consider to integrate with proof planning techniques for verification in future.

Naproche [8] is also based on an extension of DRT. Like MathNat, Naproche translates its output to first order formulas. Currently, it has a quite limited

[2] http://coq.inria.fr/
[3] http://wiki.portal.chalmers.se/agda/

controlled language without rich linguistic features. Further, like MathNat, the problem of reasoning gaps in math text is yet to be tackled.

WebALT[3] is a GF-based project that tries to make multilingual mathematical exercises in seven languages. It has a fairly good coverage for mathematical statements. However, a significant part of mathematics, i.e. the language of proof is out of scope of this work.

9 Conclusion and Future work

For textbook mathematical text, we do not know any system that provides such linguistic features as MathNat does. The coverage of CLM facilitates some common reasoning patterns found in proofs but it is still limited. So for coverage, we aim at working in two directions: enlarging the grammar manually for common patterns and supporting the ability of being extensible for some parts of CLM grammar.

Context building for symbolic expressions and equations should be improved and we want to find consistent algorithm to pick the right pronoun referents. Some theorems and their proofs from elementary number theory, set theory and analysis were parsed in CLM, and translated in MathAbs, which was further translated in equivalent first order formulas. But we were able to validate only few of them due to the reasoning gaps. So we want to improve the MathNat interaction with automated theorem provers. We also want to explore the possibility of integrating with various proof assistants.

References

1. H. Barendregt. 2003. Towards an Interactive Mathematical Proof Language. Thirty Five Years of Automath, Ed. F. Kamareddine, Kluwer, 25-36.
2. N. G. de Bruijn. 1994. Mathematical Vernacular: a Language for Mathematics with Typed Sets. In R. Nederpelt, editor, selected Papers on Automath, pages 865-935.
3. O. Caprotti. WebALT! Deliver Mathematics Everywhere. In Proceedings of SITE 2006. Orlando March 20-24, 2006.
4. M. Forsberg, A. Ranta. 2004. Tool Demonstration: BNF Converter HW'2004, ACM SIGPLAN.
5. Grammatical Framework Homepage. `http://www.grammaticalframework.org/`
6. T. Hallgren & A. Ranta. 2000. An extensible proof text editor. Springer LNCS/LNAI 1955.
7. F. Kamareddine & J. B. Wells. 2008. Computerizing Mathematical Text with MathLang, ENTCS, 205, 1571-0661, Elsevier Science Publishers B. V. Amsterdam.
8. D. Kühlwein, M. Cramer, P. Koepke, and B. Schröder. 2009. The Naproche System. In Intelligent Computer Mathematics, Springer LNCS, ISBN: 978-3-642-02613-3.
9. Per Martin-Löf. 1984. Intuitionistic Type Theory. Bibliopolis, Napoli.
10. The PhoX Proof Assistant. `http://www.lama.univ-savoie.fr/~RAFFALLI/af2.html`
11. A. Ranta. 2004. Grammatical Framework: A Type-Theoretical Grammar Formalism. Journal of Functional Programming, 14(2):145189.
12. A. Ranta. 2008. Grammars as Software Libraries. To appear in G. Huet, et al. (eds), From semantics to computer science. Cambridge University Press.
13. D.L. Simon. 1988. Checking natural language proofs, in: Springer LNCS 310.
14. P. Thévenon. 2006. PhD Thesis. Vers un assistant à la preuve en langue naturelle. Université de Savoie, France.
15. P. Thévenon. 2004. Validation of proofs using PhoX. ENTCS. `www.elsevier.nl/locate/entcs`
16. M. Wenzel. 1999. Isar: a generic interpretative approach to readable formal proof documents, Springer LNCS 1690.
17. F. Wiedijk. 2003. Formal Proof Sketches. TYPES 2003, Italy, Springer LNCS 3085, 378-393
18. C. Zinn. 2006. Supporting the formal verification of mathematical texts. Journal of Applied Logic, 4(4).