Restoring Natural Language as a Computerised Mathematics Input Method

Fairouz Kamareddine, Robert Lamar, Manuel Maarek, and J. B. Wells

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

Abstract. Methods for computerised mathematics have found little appeal among mathematicians because they call for additional skills which are not available to the typical mathematician. We herein propose to reconcile computerised mathematics to mathematicians by restoring natural language as the primary medium for mathematical authoring. Our method associates portions of text with grammatical argumentation roles and computerises the informal mathematical style of the mathematician. Typical abbreviations like the aggregation of equations $a = b > c$, are not usually accepted as input to computerised languages. We propose specific annotations to explicate the morphology of such natural language style, to accept input in this style, and to expand this input in the computer to obtain the intended representation (i.e., $a = b$ and $b > c$). We have named this method *syntax souring* in contrast to the usual *syntax sugaring*. All results have been implemented in a prototype editor developed on top of TEXMACS as a GUI for the core grammatical aspect of MathLang, a framework developed by the ULTRA group to computerise and formalise mathematics.

1 Introduction

Over several millennia, the mathematical community has developed a prodigious mass of knowledge. Effective communication of this knowledge has been essential to its dissemination. As various results have arisen and circulated, patterns and conventions have been developed for their sound and acceptable communication, leading to a *de facto* style of recording mathematical concepts in natural language. This style is sufficiently standardised to effectively communicate the most esoteric of ideas, while being flexible enough to record the variety of mathematical topics which have been explored in the academy of yesteryear and today.

1.1 State of the field

Since the advent of computer-aided proof in the 1960s, mathematicians and computer scientists have been seeking effective ways to encode mathematical concepts in languages of varying structure. Some theorem provers are highly rigid and distant from natural language, while others such as Mizar and Isar have a syntax similar to the mathematician's style. Each prover has its proponents and favoured applications, but they are all stark and restrictive when compared with

the fluidity of natural language. None currently has an infrastructure to provide a direct mapping from a typical natural language mathematical text to its own language but they all have methodologies to offer natural language integration. We group these methodologies into four categories.

1. **Proof code with embedded natural language.** In a typical formal proof language—such as Isabelle [1] or Coq [2]—there are facilities to incorporate natural language alongside formal definitions and proofs. Natural language text parts are treated as commentary in a literate proof document and omitted by the verification. This method uses *structured comments*, akin to programming languages, for generating documentation out of programming code. In a similar fashion, recent developments of intuitive text editors have permitted plugin-interfacing with theorem provers [3–5].
2. **Syntax *à la natural language*.** Formal languages often suffer from rough syntax and strict grammar. To soften the use of formal languages some efforts have been made to adapt these syntaxis and grammars to mathematicians' habits. Some developments have gone far in this direction to obtain formal proof documents that *look like* natural language texts. The main examples are Mizar [6] and Isar [7], but more recently some calculi [8, 9] were developed pursuing the idea of a formal representation for pseudo-natural language.
3. **Semantic Web data model.** Mathematical natural language is a vague and imprecise language which is unfriendly to computation. Web technologies offer a compromise in the way they encapsulate natural language and extend it with semantic tagging and hyperlinking. OMDoc [10] is a precursor in this domain.
4. **Natural language generator.** If the starting point is a formally defined language then a natural language representation of the formal content can be produced. The proof assistant HEΛM [11] has this capability. Furthermore, [12] and [13] provide facilities to personalise the natural language generated.

We conclude that the primary input for a theorem prover is generally a formal language and that the natural language of a theorem prover's document is a formalisation side effect. In case 2 the document is written in an altered and restricted natural language while in case 4 the generated (natural) text is only available *after* providing the input through a significantly restricted language. These pseudo-natural languages are by no means the only legitimate representation of mathematics. Recent work—not pertaining to any particular system—has explored the more general issue of comparing various formal representations [14], demonstrating the importance of flexibility in establishing formal models and providing concrete examples such as the formalisation of matrices [15].

1.2 Contributions

From the above it may be seen that for a semantically helpful computerisation of mathematical knowledge, today's systems require the use of a formal language which differs in some way from the common, natural, mathematical language. This paper proposes a method to **restore natural language as the primary input for computerised mathematics**. The motivation is to provide mathe-

maticians with straightforward tools they can employ to use computers in their everyday work. Efforts towards this goal fall into several categories.

1. *An integrated system for natural-language text input and grammatical categorisation.* A new approach to authoring natural language texts is presented in Section 2. As the natural language text is composed, each word or phrase is placed into a certain grammatical category as enumerated in Table 1. This is achieved by annotating the original natural language text either during or after its composition. A typical work pattern is presented in Section 2.4.
2. *Tools for reconciling complex expressions to simple grammatical categories.* In Section 3 we give several transformations a user may apply to plain text in order to cause the expression to cleanly fit a grammatical classification. These tools are built on top of the aforementioned authoring approach and work to reconcile varying natural writing styles to the stricter grammatical rules. The effect is to duplicate, shuffle, and unfold natural language text so that it is expressed in an explicit manner and strict order. These rewriting rules constitute a "dual" of syntax sugaring which we call *syntax souring*.
3. *An abstract framework to assert the foundational reliability of the proposed system.* The narrative in Section 4 presents an operational system which provides a rigorous framework upon which the denotational meaning can rest. It provides a data structure for mathematical documents, incorporating the grammatical categorisation, syntax souring notions, and a set of rewriting rules which achieve the souring functionality presented in the earlier sections.

Throughout the paper, we motivate our proposal on a supplement example of an excerpt from a textbook [16, Ch. 12] which, due to space limitation, is available as a supplement to this paper at this paper's authors' respective web pages.

1.3 Background: MathLang

In the development of computer proof aids, a major goal is to establish a correspondence between natural language mathematics and some core language (e.g., Automath, Coq, Mizar). The MathLang proposal [17] is to analyse the text in terms of various *aspects* exhibited by the document. [18] outlined the *core grammatical* aspect (CGa). CGa is concerned first with terminology, entities, and modifiers which express the knowledge and moreover their relationships to one another. Table 1 lists in **bold face**, the grammatical categories used at the CGa aspect of MathLang together with the colour coding. In the current paper we focus on a *text and symbol* aspect (TSa) of mathematical knowledge which is able to flexibly represent natural language mathematics.

term	common mathematical objects like "$a + b$" or "an additive identity 0".
set	Sets of mathematical objects such as "$\mathbb{N}$".
noun	families of **term**s such as "ring".
adjective	defines new **noun**s from old ones. E.g., "Abelian" is an **adjective** which modifies the **noun** "ring" to create the new **noun** "Abelian ring".
statement	Expressions like "$a + 0 = a$" which describe mathematical properties.
declaration	the type signature of a new **term**, **set**, **noun**, **adjective**, or **statement**.
definition	defines new symbols in mathematical texts.
step	A group of mathematical assertions.
context	preliminary assertions prior to a **step**.

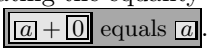
Table 1. MathLang's grammatical categories

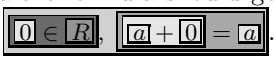
2 Box annotation, an explicit typing of expressions

We propose an authoring technique in which the mathematical text is input to the computer exactly as it is written on paper by the mathematician. As an author composes a document, it is desirable to truly derive any formal or symbolic version from this original document. We propose to decorate the original text with extra information. This extra content has to be more precise, complete and computation-friendly than natural language. With such extra information intermingled with the original text, it is possible to ensure that subsequent translations are consistent with and faithful to the natural language text.

2.1 Box annotation

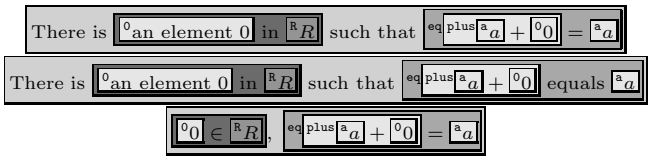
The approach of this paper augments the original natural language text with supplementary information. We do so by wrapping (at the screen), pieces of text with *annotation boxes*. The background colour of an annotation box informs about the MathLang-grammatical role of the wrapped text (following the colour coding of Table 1). Notice that once we remove these annotation boxes we find the text completely unchanged. Take from our supplement example the sentence "There is an element 0 in R such that $a+0=a$". The grammatical information (in terms of MathLang's grammatical constructions) can be easily inferred from the original text as shown by the following annotation boxes. The boxes surrounding "an element 0", "a", "0" and "$a+0$" indicate that these expressions are **terms**. "R" is wrapped in a **set** box and "an element 0 in R" in a **declaration** box. The box surrounding "$a+0=a$" indicates that this equation is a **statement**. The whole sentence is put in a **step** box.

This expression would correspond to the pseudo-logic code `eq(plus(a,0),a)` which differs from its box-annotated natural language equivalent by its namespaces. The symbol "+" corresponds to the identifier `plus`. Accordingly, one might argue that the symbols `=` and `+` could have been used with infix notation and relevant symbols' precedence. We would have obtained an expression `a+0=a` which is similar to the natural language sentence's equation. But imagine a situation where, instead of stating the equality between $a+0$ and a by an equation, the verb "equal" is used: 

. The sentence would be printed differently but would still mean that `a+0=a`. An equation and its natural language equivalent should reflect the same meaning (`a+0=a` in our example). Similarly, a natural language sentence and its equivalent formula should get similar box annotations. Our sentence could look like: 

.

2.2 Interpretation

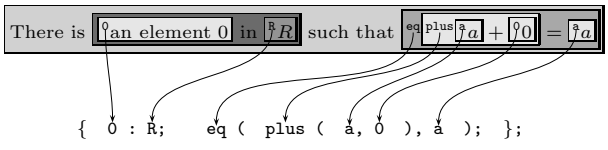


To establish the meaning of the text contained in each annotation box, we attribute to each box its interpretation in our grammar (see Table 1). The boxes

surrounding "an element 0" and "0" get `0` as interpretation attribute. The box surrounding "R", "a", "$a+0$" and the equation get as interpretation attributes `R`, `a`, `plus` and `eq`, respectively. Each interpretation attribute is printed in a typewriter typeface on the left hand side of the annotation box.
With these examples we see that MathLang's grammar is not a natural language grammar but a mathematical *justifications* grammar (following de Bruijn [19]).

2.3 Nested annotations

In our example we see also that some boxes are inside other boxes. In the case of our equation, each inner box is interpreted as an argument for its surrounding box. The nesting of boxes indicates that some annotated expressions are sub-expressions of others. It is a straightforward automatic process to create a MathLang grammatical expression out of a text with box annotations.
We show here the MathLang grammatical expression corresponding to our box-annotated text. This expression is written using the abstract syntax we presented in [18]. Note that this syntax is not meant to be used by the end-user of MathLang, it is only designed for theoretical discussion on MathLang's grammar. The MathLang end-user edits his natural language text with annotation boxes, as shown in Section 2.4 and in the supplement example. The internal syntax used in our implementations follows XML recommendations.



2.4 Automatic grammatical analysis

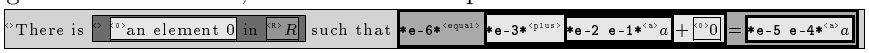
This authoring method with annotation boxes was implemented as a plugin for the scientific text editor TEXMACS. During or after the editing of a natural language text, an author is asked to wrap relevant pieces of text in MathLang's annotation boxes. Customised views are provided within the MathLang plugin to toggle the display of several features of the document including the coloured boxes resulting in this wrapping and the interpretations introduced in Section 2.2. The user easily obtains the following views (once with annotation boxes printed as coloured boxes and then with these boxes hidden):

There is an element 0 in R such that $a+0=a$

The MathLang plugin communicates the content of the document to the MathLang grammar checker given in [18], employing TEXMACS as an integrated graphical environment for natural language input, annotation, and grammar checking.

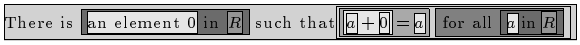
Continuing with our sentence-example, let us assume that R, 0, = and + were properly introduced in the larger document. When the user is satisfied with his annotation of the sentence, the TEXMACS plugin is instructed to send the entire document to the type checker. The checker analyses the grammatical structure of the MathLang document and finds out that a has not been properly

introduced in our sentence. A set of errors[1] with their locations in the TEXMACS document are sent back to the plugin to be shown to the user. Here is a view of the text with annotation boxes and their interpretations printed in between angle brackets < and >, and errors' labels printed in between stars *.




```
Error (e-1): Anticipated instance of "a"
Error (e-2): Categories mismatch, Unspecified expected, not term.
Error (e-3): Types mismatch for "plus", (term,term):term expected, not (Unspecified,term):term.
Error (e-4): Anticipated instance of "a"
Error (e-5): Categories mismatch, Unspecified expected, not term.
Error (e-6): Types mismatch for "equal", (term,term):stat expected,
not (term,Unspecified):stat.
```

To fix these errors we simply define a as it was done in the original text (see the supplement example). The extra "for all a in R" text is wrapped in a **context** box annotation which indicates that it forms the context of the equation.



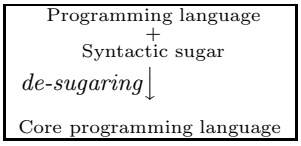
3 Souring annotation

The grammatical box annotations of Section 2 are guided by the style in which the original natural-language sentences were written. Mathematical writing styles are *uneven* and do not always fit such simplistic annotations. To adapt to any style, we need additional box annotations which help interpret the author's style. We believe it is necessary to separate grammatical and style annotations.

3.1 Syntax souring.

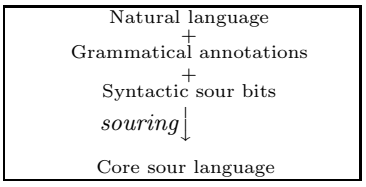
Mathematicians use mathematical natural language as a medium for communicating mathematical knowledge, but this language is highly automation-unfriendly for computer software. We showed in [12] that MathLang has constructions that correspond to the way common mathematical justifications are structured. MathLang is automation-friendly and mimics the mathematical natural language structure of justification. Therefore MathLang authoring does not require the user to alter or translate the document's knowledge for computerisation, although there is a need to adjust the writing style when encoding text directly into the core MathLang language. Because we regard our starting language, natural language, to be the *sweetest* for human readers, we call this modification *syntax souring*. This term describes the process of transforming natural language into syntactically formalised language (the core grammatical MathLang of [18]). The additives needed to describe how to perform a transformation of natural language to a core formalised language are known as *souring annotation*.

Syntax sugaring. The notion of *syntax sugaring* is well known by programmers. Syntactic sugar is added to the syntax of programming languages to make it easier to use by humans. Syntax sugaring lightens the syntax without affecting expressiveness.



[1] The high number of errors is due to the fact that the checking of the document does not stop after one error is found but analyses the entire document. An error may point at several locations in the document, this to cover all expression involved in a typing error.

Souring: dual of de-sugaring. Syntactic sugar is usually an additive for the syntax of formal language. *De-sugaring* is the process of getting rid of the sugared bits by replacing them with proper core syntax expressions.



In our case the primary input is the mathematician's natural language which we want to extend for computer software use. *Souring* unfolds the sour bits to produce a *sour document*, i.e. a document which is formal enough to be understood by computer software. The original document and the sour one do not belong to the same type of document.

The *duality* between syntax sugaring and syntax souring resides in the fact that both are methods to humanise the authoring of rigid languages but have a different starting point (i.e., programming language for syntax sugaring and natural language for syntax souring). De-sugaring adapts rigid languages for human consumption. Souring rigidifies natural language for software use.

3.2 Denotational representation

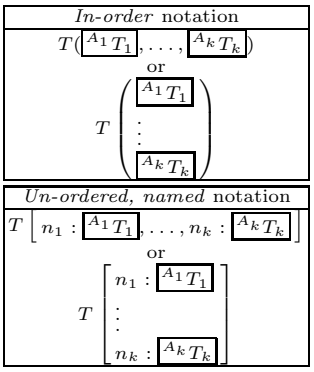
We give here the denotational representation which is formalised in Section 4.1.

Document. Our starting point is the mathematician's text (as he wrote it on paper) which is composed by a mixture of natural language text and formulae formed by symbols. This primary input corresponds to $\mathcal{D}_{\mathcal{F}}$ (formed by $\mathcal{F}$ individuals) in the abstract syntax of Section 4.1. We add to this primary input, grammatical and souring annotations that wrap portions of the text. We already saw in Section 2 how we represent grammatical annotations. In this section we explain how we represent the souring annotations discussed in Section 3.1. We denote by T a portion of text which may include formulae, grammatical annotations and souring annotations. We denote by A an arbitrary annotation.

Grammatical annotations. A grammatical annotation is an instance of one of the grammatical categories **term**, **set**, **noun**, **adjective**, **statement**, **declaration**, **definition**, **context**, or **step** (see Table 1). Each instance of a grammatical annotation may get an attribute which corresponds to the grammatical annotation's interpretation given in Section 2. We represent grammatical annotations by a box whose background colour—according to the colour coding of Table 1—informs the grammatical category and whose interpretation is printed on the upper left-hand side of the box using `courier` typeface. Here is for instance the term a annotated with a **term**-box with `"a"` as interpretation: $\boxed{{}^{\texttt{a}}a}$. We use G, G', G_1, etc., to range over grammatical interpretations. Grammatical annotations correspond to $\mathcal{G}$ labels in the formal system presented in Section 4.1.

Souring annotations. Sour bits correspond to souring annotations. We denote them by a distinguishable font colour and a thicker box for the annotation they describe (i.e., $\boxed{{}^{\text{list}}a, b, c}$). We define in the rest of this paper the following syntax souring annotations (which correspond to the elements souring labels $\mathcal{S}_u$ of Section 4.1): `position` i, `fold-right`, `fold-left`, `base`, `list`, `hook`, `loop`, `shared` and `map` (where i is a natural number).

Patterns. To describe the souring rules, we need to reason about the annotation boxes contained in a text. To do so, we add parameters to a text T to identify the text patterns that could be transformed. We use two different notations for these parametrised texts: the *in-order* notation where arguments should appear in T in the same order as they appear in the pattern and the *un-ordered* notation where the order of arguments is unimportant. We denote such parametrised notation, with $\boxed{A_1 T_1}$, $\ldots$, $\boxed{A_k T_k}$ being the arguments for T, as in the accompanying diagram. Sometimes, optional names $n_1, \ldots, n_k$ are used as markers to determine the argument's location in the text. The behaviour of parametrised text is reflected in the de-formatting function (see Definition 5) and compatibility property (see Definition 6) stated in Section 4.

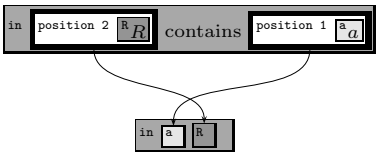


3.3 Souring transformations

In this section we indicate how to use our souring annotations and describe the result of a souring transformation where the souring notation is unfolded to obtain a text where grammatical annotations are similar to those of Section 2. Such a document could then be checked according to the MathLang grammatical checker of [18] discussed briefly in Section 2.4.

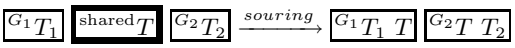
Re-ordering. position i When dealing with a natural language mathematical text, one regularly faces situations where two expressions holding similar knowledge are ordered differently. The re-ordering transformation corresponds to $\rightarrow_{pos}$ of Section 4.2. Considering the expression "a in R" from our supplement example, one can easily imagine the author using "R contains a" instead. The `position` souring annotation is meant for reordering inner-annotations. The souring rewriting function reorders the elements according to their position indices.

The expressions "a in R" and "R contains a" should both be interpreted as `in(a,R)` if `in` is the set membership relation. To indicate in the second expression that the order of the argument is not the "reading" order, we annotate R and a with `position 2` and `position 1`, respectively. It is common for binary symbols like $\subset$ to have a mirror twin like $\supset$. The `position` souring annotation usefully gives the same interpretation to twin symbols.

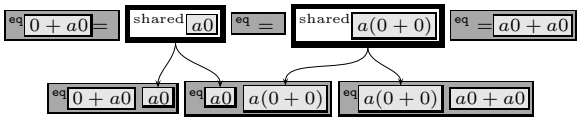


Sharing/chaining. shared hook loop Mathematicians have the habit of aggregating equations which follow one another. This creates reading difficulties for novices yet contributes to the aesthetic of mathematical writing. The `shared` and `hook`/`loop` souring annotations are solutions which elucidate such expressions.

The `shared` annotation indicates that an expression is to be used by both its preceding and following expressions. The shared expression is inlined at the end of the preceding expression and at the beginning of the following one. This transformation corresponds to $\twoheadrightarrow_{share}$ of Section 4.2.



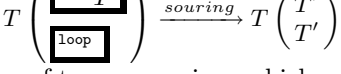
The document example we chose to computerise (see our supplement example) contains several sentences which are made easier to computerise by the use of sharing. The multiple equation "0+a0 = a0 = a(0+0) = a0+a0" is certainly the best example as it requires the use of two `shared` annotations. We can see that $a0$ and $a(0+0)$ are shared by two equations each. We annotate them as being shared to obtain an unfolded result equivalent to "0+a0 = $a0$, a0 = a(0+0), a(0+0) = a0+a0".

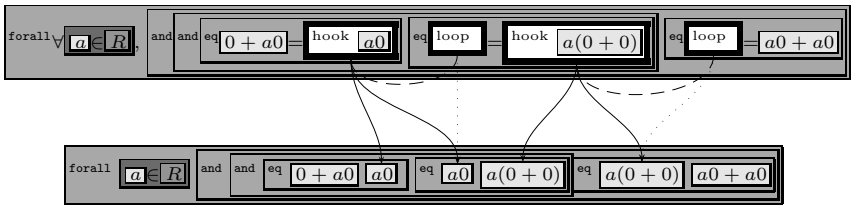


The full interpretation of this expression being:

```
eq(plus(0,times(a,0)),times(a,0));
eq(times(a,0),times(a,plus(0,0)));
eq(times(a,plus(0,0)),plus(times(a,0),times(a,0)))
```

The tuple of souring annotations `hook`/`loop` indicates the expression contained in the hook should be repeated in the loop. We named this concept chaining because it permits the separation of two expressions which are effectively printed as one in a natural language text. Chaining provides results similar to sharing (any sharing could be expressed in terms of chaining), but is more expressive. This transformation corresponds to $\twoheadrightarrow_{chain}$ of Section 4.2.

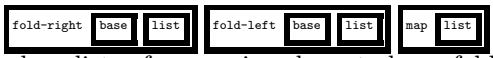




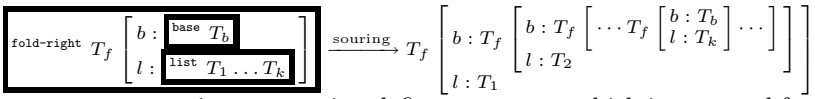
The full interpretation of this expression being:

```
forall(a:R, and( and( eq(plus(0,times(a,0)),times(a,0)),
                      eq(times(a,0),times(a,plus(0,0))) ),
                eq(times(a,plus(0,0)),plus(times(a,0),times(a,0))) ))
```

Let us see an example where a `shared` souring annotation could not have been used. If we consider the equation we used in the sharing example and decide to quantify this equation over a, we would obtain "$\forall a \in R$, 0+a0 = a0 = a(0+0) = a0+a0" which is effectively a shortcut for "$\forall a \in R$, 0+a0=a0 $\wedge$ a0=a(0+0) $\wedge$ a(0+0)=a0+a0". We can see that in this example the individual equations are combined using two binary operators `and`, the combination of whose annotation boxes disallows the use of `shared`.

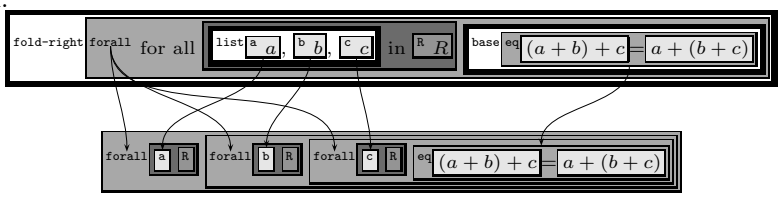
List manipulations. 

The list souring annotations indicate how lists of expressions have to be unfolded into MathLang interpretations. We define two list folding annotations, `fold-right` and `fold-left`, and a mapping annotation, `map`.



The `fold-right` souring annotation defines a pattern which is repeated for each element of the list argument. For each repeated pattern, the `list` inner annotation is replaced by one element of the list and the `base` inner annotation is replaced by the pattern with the next element of the list. `fold-left` works similarly but starts with the last element of the list. These transformations correspond to $\rightarrow_{fold}$ of Section 4.2.

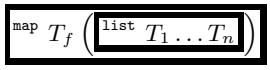
A major use of the `fold-right` souring annotation is to handle quantification over multiple variables. Considering the sentence "for all a, b, c in R *[...]* $(a+b)+c = a+(b+c)$", we would like to use one single `forall` instance for each variable a, b and c. We simply annotate the list of variables as such and the base equation as `base` and the souring unfolding creates a fully expanded interpretation on our behalf.



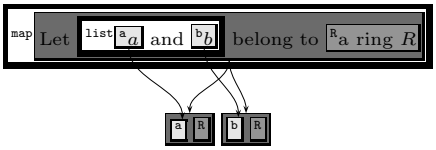
The full interpretation of this expression being:

```
forall(a:R, forall(b:R, forall(c:R,eq(plus(plus(a,b),c),plus(a,plus(b,c))) ) ) )
```

The `map` souring annotation also defines a pattern but with only one argument being `list`. This pattern is also repeated for each element of the list. The resulting expression is a sequence. It corresponds to $\rightarrow_{map}$ defined in Section 4.2.



Similarly to folding, this souring annotation is useful for declarations, definitions or statements over several things. In the case of the sentence "Let a and b belong to a ring R" taken from our supplement example, the variables a and b are declared simultaneously.



4 Operational system

Having presented our method in an intuitive, denotational style, we now give the formal system behind it and the foundation for MathLang documents.

4.1 Abstract syntax

Let $\mathbb{N}$ denote the natural numbers, use $(-;-)$ to denote ordered pairs, and let functions be sets φ of ordered pairs with a domain $\mathrm{dom}(\varphi) = \{a \mid \exists b \text{ such that } (a;b) \in \varphi\}$. A sequence is a function s for which $\mathrm{dom}(s) = \{n \mid 0 \leq n < k\}$ for some $k \in \mathbb{N}$. We write [] for the empty sequence and $[x_0, x_1, \ldots, x_n]$ for the sequence s such that $s(i) = x_i$ for each $i \in \mathrm{dom}(s) = \{0, \ldots, n\}$. Upon that

sequence is defined the metric $|s| = n + 1$. We define s_1, s_2, the concatenation of sequences s_1 and s_2, as the new sequence s such that $\mathrm{dom}(s_1, s_2) = \{0, \ldots, |s_1| + |s_2| - 1\}$, $s(i) = s_1(i)$ for $i \in \mathrm{dom}(s_1)$ and $s(i) = s_2(i)$ for $i - |s_1| \in \mathrm{dom}(s_2)$. Concatenation is associative. Moreover, $[], s = s$ and $s, [] = s$.

Let $\mathcal{L} = \mathcal{F} \cup \mathcal{G} \cup \mathcal{S}$ to be the set of labels over which ℓ ranges where elements of $\mathcal{F}$, resp. $\mathcal{G}$, resp. $\mathcal{S}$, are formatting, resp. grammatical, resp. souring labels.

$\mathcal{F}$ (over which f ranges, cf. Definition 4) consists of *formatting instructions* and varies according to the typesetting system used.

$\mathcal{G} = \mathcal{C} \times \mathcal{I}$ where $\mathcal{C} = \{\mathbf{term}, \mathbf{set}, \mathbf{noun}, \mathbf{adj}, \mathbf{stat}, \mathbf{decl}, \mathbf{defn}, \mathbf{step}, \mathbf{cont}\}$, and contains identifiers for the primitive grammatical categories of Table 1. The set $\mathcal{I}$ consists of strings used for identifying abstract interpretations (e.g., `0`, `R`, `eq`, `plus` and `a` are the interpretation strings used in the examples throughout Section 2). We let g, c and i range respectively over $\mathcal{G}$, $\mathcal{C}$ and $\mathcal{I}$.

We let s range over $\mathcal{S} = \mathcal{S}_u \cup \mathcal{S}_i$ where $\mathcal{S}_u$ contains *souring identifiers* to be employed directly by the user while $\mathcal{S}_i$ holds several identifiers used internally for rewriting. $\mathcal{S}_u$ and $\mathcal{S}_i$ are disjoint and are as follows:

$\mathcal{S}_u = \{\texttt{fold-left}, \texttt{fold-right}, \texttt{map}, \texttt{base}, \texttt{list}, \texttt{hook}, \texttt{loop}, \texttt{shared}\} \cup (\{\texttt{position}\} \times \mathbb{N})$

$\mathcal{S}_i = \{\texttt{hook-travel}, \texttt{head}, \texttt{tail}, \texttt{daeh}, \texttt{liat}, \texttt{right-travel}, \texttt{left-travel}\} \cup (\{\texttt{cursor}\} \times \mathbb{N})$

Definition 1 (Document). *Let $\mathcal{D}$ be the smallest set such that:*
1. $[] \in \mathcal{D}$,
2. if $d \in \mathcal{D}$ and $\ell \in \mathcal{L}$ then $[(\ell; d)] \in \mathcal{D}$, and
3. if both d_1 and d_2 are elements of $\mathcal{D}$ then $(d_1, d_2) \in \mathcal{D}$.
A MathLang document is an element of the set $\mathcal{D}$. In addition, we denote by $\mathcal{D}_{\mathcal{F}}$, $\mathcal{D}_{\mathcal{G}}$, $\mathcal{D}_{\mathcal{F}\cup\mathcal{G}}$, $\mathcal{D}_{\mathcal{G}\cup\mathcal{S}}$ and $\mathcal{D}_{\mathcal{F}\cup\mathcal{G}\cup\mathcal{S}}$ the sets of documents for which labels are in $\mathcal{F}$, $\mathcal{G}$, $\mathcal{F} \cup \mathcal{G}$, $\mathcal{G} \cup \mathcal{S}$ and $\mathcal{F} \cup \mathcal{G} \cup \mathcal{S}$, respectively.

Remark 1 (Notational convention). For convenience, $[(\ell; d)]$ abbreviates to $\ell\langle d\rangle$. Furthermore, when not ambiguous $\ell\langle[]\rangle$ abbreviates to ℓ. In the case of grammatical labels ordered pairs, we denote the interpretation (second element of the pair) by an adjoined superscript. A pair $(c; i)$ from $\mathcal{G}$ is denoted by c^i. Similarly, an ordered pair from $\{\texttt{position}\} \times \mathbb{N}$ (respectively $\{\texttt{cursor}\} \times \mathbb{N}$) is denoted by a superscript number (second element of the pair) adjoined to `position` (respectively `cursor`).

Definition 2 (Sub-document). *We define* sub-document*, and we denote by $\sqsubset_{\mathcal{G}}$, the binary relation between documents such that:*

$$d \sqsubset_{\mathcal{G}} d \tag{SUB1}$$

$$d \sqsubset_{\mathcal{G}} g\langle d_1\rangle \text{ if } d \sqsubset_{\mathcal{G}} d_1 \tag{SUB2}$$

$$d \sqsubset_{\mathcal{G}} (d_1, d_2) \text{ if } d \sqsubset_{\mathcal{G}} d_1 \text{ or } d \sqsubset_{\mathcal{G}} d_2 \tag{SUB3}$$

Remark 2. It is important to notice that our sub-document property (SUB2) is restricted to grammatical labels which means that for any label $\ell \notin \mathcal{G}$ and any documents d_1 and d_2 such that $d_1 \sqsubset_{\mathcal{G}} d_2$, we have that $d_1 \not\sqsubset_{\mathcal{G}} \ell\langle d_2\rangle$.

Definition 3 (Label inclusion). *We define* label inclusion*, and we denote by $\widetilde{\in}_{\mathcal{G}}$, the binary relation between a label and a document such that:*

$$\ell \mathrel{\widetilde{\in}_{\mathcal{G}}} \ell\langle d\rangle \tag{INC1}$$

$$\ell \mathrel{\widetilde{\in}_{\mathcal{G}}} g\langle d\rangle \text{ if } \ell \neq g \text{ and } \ell \mathrel{\widetilde{\in}_{\mathcal{G}}} d \tag{INC2}$$

$$\ell \mathrel{\widetilde{\in}_{\mathcal{G}}} (d_1, d_2) \text{ if } \ell \mathrel{\widetilde{\in}_{\mathcal{G}}} d_1 \text{ or } \ell \mathrel{\widetilde{\in}_{\mathcal{G}}} d_2 \tag{INC3}$$

Remark 3. Note that our label inclusion property (INC2) is restricted to grammatical labels, which means that for any labels $\ell_1 \notin \mathcal{G}$ and $\ell_2 \in \mathcal{L}$ such that $\ell_1 \neq \ell_2$, and any document d such that $\ell_2 \widetilde{\in}_{\mathcal{G}} d$, we have that $\ell_2 \widetilde{\notin}_{\mathcal{G}} \ell_1\langle d\rangle$.

Definition 4 (Rendering functions). *Let* $\mathrm{f} : \mathcal{D} \to \mathcal{D}_{\mathcal{F}}$ *be a function where:*

$$\mathrm{f}([]) = [] \tag{FORM1}$$

$$\mathrm{f}(\ell\langle d\rangle) = \begin{cases} \ell\langle \mathrm{f}(d)\rangle & \textit{if } \ell \in \mathcal{F} \\ \mathrm{f}(d) & \textit{otherwise} \end{cases} \tag{FORM2}$$

$$\mathrm{f}(d_1, d_2) = \mathrm{f}(d_1), \mathrm{f}(d_2) \tag{FORM3}$$

Thus, f *flattens a given document* d *at any label from* $\mathcal{G}$ *or* $\mathcal{S}$*, removing all such labels. Once this is achieved, it will be possible to use* $\mathrm{r} : \mathcal{D}_{\mathcal{F}} \to \mathcal{F}$*, where:*

$$\mathrm{r}([]) = \varepsilon \tag{REN1}$$

$$\mathrm{r}(f\langle d\rangle) = \mathrm{fill}(f, [\mathrm{r}(d(0)), \ldots, \mathrm{r}(d(|d|-1))]) \tag{REN2}$$

$$\mathrm{r}(d_1, d_2) = \mathrm{r}(d_1) \bullet \mathrm{r}(d_2) \tag{REN3}$$

Where, in a specific typesetting system, ε *is the blank formatting instruction,* $\bullet$ *is the composition operator and* fill *is a formatting-system-specific function. The function* fill *interprets a formatting instruction (first argument) with a sequence of rendered documents passed as argument. One can imagine a formatting instruction to be a template with holes and* fill *to simply fill these holes. The number of vacancies exhibited by the first argument of* fill *should be equal to the length of the sequence, which is the second argument of* fill*. The function* fill *returns an element of the set* $\mathcal{F}$ *which is a formatting instruction requiring no argument.*

Definition 5 (De-formatting function). *To prepare a document for souring, we strip it of all formatting elements using the function* $\mathrm{df} : \mathcal{D} \to \mathcal{D}_{\mathcal{G}\cup\mathcal{S}}$ *where:*

$$\mathrm{df}([]) = [] \tag{DF1}$$

$$\mathrm{df}(\ell\langle d\rangle) = \begin{cases} d & \textit{if } \ell \in \mathcal{F} \\ \ell\langle \mathrm{df}(d)\rangle & \textit{otherwise} \end{cases} \tag{DF2}$$

$$\mathrm{df}(d_1, d_2) = \mathrm{df}(d_1), \mathrm{df}(d_2) \tag{DF3}$$

4.2 Souring rewriting rules

Definition 6 (Compatibility, Reflexive transitive closure, Normal form). *We define the following compatibility property for a rewriting rule* $\to_n$*.*

$$d_1, d, d_2 \to_n d_1, d', d_2 \textit{ if } d \to_n d' \tag{COMP1}$$

$$g\langle d\rangle \to g\langle d'\rangle \textit{ if } d \to_n d' \tag{COMP2}$$

We denote by $\twoheadrightarrow_n$ *the reflexive transitive closure of* $\to_n$*.*
We define the n*-normal form relatively to* $\to_n$ *and we denote by* NF_n *the property on a document* d *such that no* $\twoheadrightarrow_n$ *rewriting can be applied to* d*.*
Note that our compatibility rule (COMP2) *is restricted to grammatical labels.*

Below are the formal rewriting rules for souring transformations from Section 3.1.

$$\texttt{head}\langle d_1,d_2\rangle \to_{list} d_1,\texttt{head}\langle d_2\rangle \text{ where } \texttt{list} \,\widetilde{\not\in}_{\mathcal{G}}\, d_1$$
$$\texttt{tail}\langle d_1,d_2\rangle \to_{list} d_1,\texttt{tail}\langle d_2\rangle \text{ where } \texttt{list} \,\widetilde{\not\in}_{\mathcal{G}}\, d_1$$
$$\texttt{daeh}\langle d_1,d_2\rangle \to_{list} d_1,\texttt{daeh}\langle d_2\rangle \text{ where } \texttt{list} \,\widetilde{\not\in}_{\mathcal{G}}\, d_1$$
$$\texttt{liat}\langle d_1,d_2\rangle \to_{list} d_1,\texttt{liat}\langle d_2\rangle \text{ where } \texttt{list} \,\widetilde{\not\in}_{\mathcal{G}}\, d_1$$
$$\texttt{head}\langle g\langle d_1\rangle,d_2\rangle \to_{list} g\langle\texttt{head}\langle d_1\rangle\rangle,d_2 \text{ where } \texttt{list} \,\widetilde{\in}_{\mathcal{G}}\, d_1$$
$$\texttt{tail}\langle g\langle d_1\rangle,d_2\rangle \to_{list} g\langle\texttt{tail}\langle d_1\rangle\rangle,d_2 \text{ where } \texttt{list} \,\widetilde{\in}_{\mathcal{G}}\, d_1$$
$$\texttt{daeh}\langle g\langle d_1\rangle,d_2\rangle \to_{list} g\langle\texttt{daeh}\langle d_1\rangle\rangle,d_2 \text{ where } \texttt{list} \,\widetilde{\in}_{\mathcal{G}}\, d_1$$
$$\texttt{liat}\langle g\langle d_1\rangle,d_2\rangle \to_{list} g\langle\texttt{liat}\langle d_1\rangle\rangle,d_2 \text{ where } \texttt{list} \,\widetilde{\in}_{\mathcal{G}}\, d_1$$
$$\texttt{head}\langle\texttt{list}\langle g\langle d_1\rangle,d_2\rangle,d_3\rangle \to_{list} g\langle d_1\rangle,d_3$$
$$\texttt{tail}\langle\texttt{list}\langle g\langle d_1\rangle,d_2\rangle,d_3\rangle \to_{list} d_2,d_3$$
$$\texttt{daeh}\langle\texttt{list}\langle d_1,g\langle d_2\rangle\rangle,d_3\rangle \to_{list} g\langle d_2\rangle,d_3$$
$$\texttt{liat}\langle\texttt{list}\langle d_1,g\langle d_2\rangle\rangle,d_3\rangle \to_{list} d_1,d_3$$

$$g_1\langle d_1\rangle,\texttt{shared}\langle d\rangle,g_2\langle d_2\rangle \to_{share} g_1\langle d_1,d\rangle,g_2\langle d,d_2\rangle$$

$$\texttt{hook}\langle d\rangle \to_{chain} d,\texttt{hook-travel}\langle d\rangle$$
$$\texttt{hook-travel}\langle d\rangle,\texttt{loop} \to_{chain} d$$
$$\texttt{hook-travel}\langle d_0\rangle,d_1,d_2 \to_{chain} d_1,\texttt{hook-travel}\langle d_0\rangle,d_2 \text{ where } \texttt{loop} \,\widetilde{\not\in}_{\mathcal{G}}\, d_1$$
$$\texttt{hook-travel}\langle d_0\rangle,g\langle d_1\rangle \to_{chain} g\langle\texttt{hook-travel}\langle d_0\rangle,d_1\rangle$$
$$g\langle d_1,\texttt{hook-travel}\langle d_0\rangle\rangle \to_{chain} g\langle d_1\rangle,\texttt{hook-travel}\langle d_0\rangle$$

$$\texttt{position}^i\langle d_1\rangle,\texttt{position}^j\langle d_2\rangle \to_{pos} \texttt{position}^j\langle d_2\rangle,\texttt{position}^i\langle d_1\rangle \text{ where } j<i$$
$$\ell\langle\texttt{position}^1\langle d_1\rangle,d_2\rangle \to_{pos} \ell\langle d_1,\texttt{cursor}^1,d_2\rangle$$
$$\texttt{cursor}^i,\texttt{position}^{i+1}\langle d_1\rangle,d_2 \to_{pos} d_1,\texttt{cursor}^{i+1},d_2$$
$$\ell\langle d,\texttt{cursor}^i\rangle \to_{pos} \ell\langle d\rangle$$

$$\texttt{fold-right}\langle d_0\rangle \to_{fold} \texttt{right-travel}\langle d_2\rangle,d_1 \text{ where } d_0 \twoheadrightarrow_{souring} d_0',\ \texttt{head}\langle d_0'\rangle \twoheadrightarrow_{list} d_1 \text{ and } \texttt{tail}\langle d_0'\rangle \twoheadrightarrow_{list} d_2$$
$$\texttt{right-travel}\langle d_1,d_2\rangle \to_{fold} d_1,\texttt{right-travel}\langle d_2\rangle \text{ where } \texttt{base} \,\widetilde{\not\in}_{\mathcal{G}}\, d_1$$
$$\texttt{right-travel}\langle g\langle d_1\rangle,d_2\rangle \to_{fold} g\langle\texttt{right-travel}\langle d_1\rangle\rangle,d_2 \text{ where } g{\neq}\texttt{base} \text{ and } \texttt{base} \,\widetilde{\in}_{\mathcal{G}}\, d_1$$
$$\texttt{right-travel}\langle d_1\rangle,\texttt{base}\langle d_2\rangle \to_{fold} d_2,\texttt{right-travel}\langle d_2\rangle \text{ where } \texttt{list} \sqsubset_{\mathcal{G}} d_1$$
$$\texttt{right-travel}\langle d_1\rangle,\texttt{base}\langle d_2\rangle \to_{fold} \texttt{fold-right}\langle d_1\rangle$$

$$\texttt{fold-left}\langle d_0\rangle \to_{fold} \texttt{left-travel}\langle d_2\rangle,d_1 \text{ where } d_0 \twoheadrightarrow_{souring} d_0',\ \texttt{daeh}\langle d_0'\rangle \twoheadrightarrow_{list} d_1 \text{ and } \texttt{liat}\langle d_0'\rangle \twoheadrightarrow_{list} d_2$$
$$\texttt{left-travel}\langle d_1,d_2\rangle \to_{fold} d_1,\texttt{left-travel}\langle d_2\rangle \text{ where } \texttt{base} \,\widetilde{\not\in}_{\mathcal{G}}\, d_1$$
$$\texttt{left-travel}\langle g\langle d_1\rangle,d_2\rangle \to_{fold} g\langle\texttt{left-travel}\langle d_1\rangle\rangle,d_2 \text{ where } g{\neq}\texttt{base} \text{ and } \texttt{base} \,\widetilde{\in}_{\mathcal{G}}\, d_1$$
$$\texttt{left-travel}\langle d_1\rangle,\texttt{base}\langle d_2\rangle \to_{fold} d_2,\texttt{left-travel}\langle d_2\rangle \text{ where } \texttt{list} \sqsubset_{\mathcal{G}} d_1$$
$$\texttt{left-travel}\langle d_1\rangle,\texttt{base}\langle d_2\rangle \to_{fold} \texttt{fold-left}\langle d_1\rangle$$

$$\texttt{map}\langle d\rangle \to_{map} [] \text{ where } \texttt{list} \sqsubset_{\mathcal{G}} d$$
$$\texttt{map}\langle d_0\rangle \to_{map} d_1,\texttt{map}\langle d_2\rangle \text{ where } d_0 \twoheadrightarrow_{souring} d_0',\ \texttt{head}\langle d_0'\rangle \twoheadrightarrow_{list} d_1 \text{ and } \texttt{tail}\langle d_0'\rangle \twoheadrightarrow_{list} d_2$$

Definition 7 (Souring rewriting rule). *The souring rewriting rule, denoted by* $\to_{souring}$ *is defined as* $d_0 \to_{souring} d_4$ *where* $d_0 \twoheadrightarrow_{share} d_1$ *(*d_1 *being in a* NF_{share}*),* $d_1 \twoheadrightarrow_{chain} d_2$ *(*d_2 *being in a* NF_{chain}*),*$d_2 \twoheadrightarrow_{pos} d_3$ *(*d_3 *being in a* NF_{pos}*),* $d_3 \twoheadrightarrow_{lists} d_4$ *(*d_4 *being in a* NF_{lists}*).*
The souring a document is the application of $\twoheadrightarrow_{souring}$ until $NF_{souring}$ is reached.

5 Related work

The natural-to-abstract work pattern which has been presented in this paper will be useful in a wide variety of settings. One possible area of application is work being done with optical character recognition of mathematics. In the work of the Infty Project [20, 21], for example, it is desirable to automate the process of extracting information from printed material. As MathLang becomes capable of being automated, it will provide further aid to extracting semantic information from a document with as little hand-translation as possible.

The TEXMACS plugin environment and method of editing causes MathLang to be a *visual language*. Using visual languages for knowledge representation is becoming more popular, and its benefits are obvious. By displaying and editing

the logical structure of a mathematical document, the categorisation of various portions of text is made more clear and the structure more lucid. This could certainly lead to a new generation of literate programming [22].

In Computational linguistics, transformational grammars [23, Ch.5] provide a morphism method similar to souring. Nevertheless they are a natural language grammar and do not provide this separation between the original human-medium (natural language) and the software-medium (MathLang core language).

6 Conclusion and future work

We demonstrate in this paper the feasibility of restoring natural language as the primary input for mathematical authoring on computers. This method will benefit mathematicians as it permits the use of computer-assisted authoring without requiring skills in computer-based formalisation. Thus, this method will benefit the mathematical knowledge community as it makes the bridge between traditional and computerised mathematics. Since the souring rewriting rules are defined on top of a generic document format, it should be straightforward to adapt the rules to some specific formatting system and core "sour" language.

The work in [12] established a language and a system for encoding mathematical texts with transformations which permitted viewing the document in various useful forms, especially natural language. The current development improves on this by allowing the user to work directly in natural language while making use of a number of automated features to do the rest. Even so, the set of souring rules with which the system has been augmented is almost certainly incomplete. There are some known mathematical constructs (mentioned below) for which a satisfactory annotation has not yet been found, and there are surely others which have not yet come to the attention of this development team.

Current known shortcomings in the system include good handling of expressions with omitted terms, such as $\overbrace{x+\ldots+x}^{n \text{ times}}$, $2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$, and $\frac{1}{1+\frac{1}{1+\cdots}}$. Proper treatment of proof by induction is under active investigation, as well as satisfactory treatment of relations such as modular equivalence e.g., $-1 \equiv 2(\mod 3)$. Other priorities are developing/integrating methods to automate the annotation process, which at present can be redundant and tedious, and gathering data on use and work patterns by mathematicians to guide further tool and interface development. Anecdotal tests have indicated that the system is very easy to use, but further investigation will be necessary to ensure that the present—or any future—implementation provides maximal assistance to the mathematical user.

References

1. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Volume 2283 of LNCS. Springer-Verlag (2002)
2. LogiCal Project, INRIA Rocquencourt, France: The Coq Proof Assistant Reference Manual – Version 8.0. (2004) At `ftp://ftp.inria.fr/INRIA/coq/V8.0/doc/`.
3. Audebaud, P., Rideau, L.: TeXmacs as authoring tool for publication and dissemination of formal developments. Volume 103 of ENTCS., Rome (2003) 27–48

4. Autexier, S., Benzmüller, C., Fiedler, A., Lesourd, H.: Integrating proof assistants as reasoning and verification tools into a scientific WYSIWIG editor. In: User Interfaces for Theorem Provers (UITP '05) [Workshop], Edinburgh (2005)
5. Mamane, L.E., Geuvers, H.: A document-oriented Coq plugin for TeXmacs. In: Mathematical User-Interfaces Workshop 2006 [Workshop], Workingham (2006)
6. Rudnicki, P.: An overview of the Mizar project. In: Proceedings of the 1992 Workshop on Types for Proofs and Programs. (1992)
7. Wenzel, M.: Isar – a generic interpretative approach to readable formal proof documents. In: Theorem Proving in Higher Order Logics: 12th Int'l Conf., Proceedings. Volume 1690 of Lecture Notes in Computer Science., Springer (1999) 167–184
8. Autexier, S., Sacerdoti Coen, C.: A formal correspondence between OMDoc with alternative proofs and the $\bar{\lambda}\mu\tilde{\mu}$-calculus. [24] 67–81
9. Brown, C.E.: Verifying and invalidating textbook proofs using Scunak. [24] 110–123
10. Kohlhase, M.: An open markup format for mathematical documents, OMDoc (Version 1.2). Volume 4180 of Lecture Notes in Artificial Intelligence. Springer (2006)
11. Asperti, A., Padovani, L., Sacerdoti Coen, C., Schena, I.: HELM and the semantic math-web. In: Volume 2152 of LNCS, Springer (2001) 59–74
12. Kamareddine, F., Maarek, M., Wells, J.B.: Flexible encoding of mathematics on the computer. In: Mathematical Knowledge Management, 3rd Int'l Conf., Proceedings. Volume 3119 of Lecture Notes in Computer Science., Springer (2004) 160–174
13. Padovani, L., Zacchiroli, S.: From notation to semantics: There and back again. [24] 194–207
14. Kerber, M., Pollet, M.: A tough nut for mathematical knowledge management. In Kohlhase, M., ed.: Mathematical Knowledge Management – 4th International Conference, MKM 2005, Bremen, Germany, Springer, LNAI 3863 (2006) 81–95
15. Pollet, M., Sorge, V., Kerber, M.: Intuitive and formal representations: The case of matrices. In Asperti, A., Bancerek, G., Trybulec, A., eds.: Mathematical Knowledge Management. Third International Conference, MKM, Białowieza, Poland, September 19-21, Springer, LNCS 3119 (2004)
16. Gallian, J.A.: Contemporary Abstract Algebra. 5th edn. Houghton Mifflin Company (2002)
17. Kamareddine, F., Wells, J.: MathLang: A new language for mathematics, logic, and proof checking. A research proposal to UK funding body (2001)
18. Kamareddine, F., Maarek, M., Wells, J.B.: Toward an object-oriented structure for mathematical text. In Kohlhase, M., ed.: Mathematical Knowledge Management, 4th Int'l Conf., Proceedings. Volume 3863 of Lecture Notes in Artificial Intelligence., Springer (2006) 217–233
19. de Bruijn, N.G.: Checking mathematics with computer assistance. Notices of the American Mathematical Society **38** (1991) 8–15
20. Kanahori, T., Sexton, A., Sorge, V., Suzuki, M.: Capturing abstract matrices from paper. [24] 124–138
21. Raja, A., Rayner, M., Sexton, A., Sorge, V.: Towards a parser for mathematical formula recognition. [24] 139–151
22. Knuth, D.E.: Literate programming. The Computer Journal **27** (1984) 97–111
23. Farrell, P.: Grammatical Relations. Oxford Surveys in Syntax and Morphology. Oxford Linguistics (2005)
24. Borwein, J.M., Farmer, W.M., eds.: Mathematical Knowledge Management, 5th Int'l Conf., Proceedings. Volume 4108 of LNCS, Springer (2006)