Motivations for MathLang

Fairouz Kamareddine[*]

Monday 14 November 2005

[*]This talk is based on joint work with Laan and Nederpelt (see [1, 2]) and Maarek and Wells (see [3, 4])

The Goal: Open borders between mathematics, logic and computation

- Ordinary mathematicians *avoid* formal mathematical logic.
- Ordinary mathematicians *avoid* proof checking (via a computer).
- Ordinary mathematicians *may use* a computer for computation: there are over 1 million people who use Mathematica (including linguists, engineers, etc.).
- Mathematicians may also use other computer forms like Maple, LaTeX, etc.
- But we are not interested in only *libraries* or *computation* or *text editing*.
- We want *freedeom of movement* between mathematics, logic and computation.
- At every stage, we must have *the choice* of the level of formalilty and the depth of computation.

Common Mathematical Language of mathematicians: CML

+ CML is *expressive*: it has linguistic categories like *proofs* and *theorems*.
+ CML has been refined by intensive use and is rooted in *long traditions*.
+ CML is *approved* by most mathematicians as a communication medium.
+ CML *accommodates many branches* of mathematics, and is adaptable to new ones.
– Since CML is based on natural language, it is *informal* and *ambiguous*.
– CML is *incomplete:* Much is left implicit, appealing to the reader's intuition.
– CML is *poorly organised:* In a CML text, many structural aspects are omitted.
– CML is *automation-unfriendly:* A CML text is a plain text and cannot be easily automated.

A CML-text

From chapter 1, § 2 of E. Landau's *Foundations of Analysis* [Lan51].

Theorem 6. [Commutative Law of Addition]

$$x + y = y + x.$$

Proof Fix y, and let $\mathfrak{M}$ be the set of all x for which the assertion holds.
I) We have

$$y + 1 = y',$$

and furthermore, by the construction in the proof of Theorem 4,

$$1 + y = y',$$

so that

$$1 + y = y + 1$$

and 1 belongs to $\mathfrak{M}$.
II) If x belongs to $\mathfrak{M}$, then

$$x + y = y + x,$$

Therefore

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

By the construction in the proof of Theorem 4, we have

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

hence

$$x' + y = y + x',$$

so that x' belongs to $\mathfrak{M}$. The assertion therefore holds for all x. □

What are the options for computerization?

Computers can handle mathematical text at various levels:

- Images of pages may be stored. While useful, this is not a good representation of *language* or *knowledge*.
- Typesetting systems like LaTeX can be used.
- Document representations like OMDoc can be used.
- Formal logics used by theorem provers can be used.

We are gradually developing a system named MathLang which we hope will eventually allow building a bridge between the latter 3 levels.

This talk aims at discussing the motivations rather than the details.

The issues with typesetting systems

+ A system like LaTeX provides good defaults for visual appearance, while allowing fine control when needed.

+ LaTeX supports commonly needed document structures, while allowing custom structures to be created.

– Unless the mathematician is amazingly disciplined, the logical structure of symbolic formulas is not represented at all.

– The logical structure of mathematics as embedded in natural language text is not represented. Automated discovery of the semantics of natural language text is still too primitive and requires human oversight.

LaTeX example

draft documents	✓
public documents	✓
computations and proofs	✗

```
\begin{theorem}[Commutative Law of Addition]\label{theorem:6}
  $$x+y=y+x.$$
\end{theorem}
\begin{proof}
  Fix $y$, and $\mathfrak{M}$ be the set of all $x$ for which the
  assertion holds.
  \begin{enumerate}
  \item We have $$y+1=y',$$
    and furthermore, by the construction in
    the proof of Theorem~\ref{theorem:4}, $$1+y=y',$$
    so that $$1+y=y+1$$
    and $1$ belongs to $\mathfrak{M}$.
  \item If $x$ belongs to $\mathfrak{M}$, then $$x+y=y+x,$$
    Therefore
    $$(x+y)'=(y+x)'=y+x'.$$
    By the construction in the proof of
    Theorem~\ref{theorem:4}, we have $$x'+y=(x+y)',$$
    hence
    $$x'+y=y+x',$$
    so that $x'$ belongs to $\mathfrak{M}$.
  \end{enumerate}
  The assertion therefore holds for all $x$.
\end{proof}
```

The differences of OMDoc

OMDoc attempts to solve some of the difficulties of typesetting systems.

+ Translation to LaTeX (still needed) or MathML can handle visual appearance.
– Precise appearance control must work *through* a translation (difficult!).
+ OMDoc supports commonly needed document structures.
+ The tree structure of symbolic formulas is represented.
– The semantics of symbolic formulas is not represented.
– Type checking symbolic formulas (beyond arity) must be outside OMDoc.
– The logical structure of mathematics as embedded in natural language text is still not represented. There are ways to associate symbolic formulas with natural language text, but no way to check their consistency.

The beginnings of computerized formalization

- In 1967 the famous mathematician de Bruijn began work on logical languages for complete books of mathematics that can be *fully* checked by machine.
- People are prone to error, so if a machine can do proof checking, we expect fewer errors.
- Most mathematicians doubted de Bruijn could achieve success, and computer scientists had no interest at all.
- However, he persevered and built *Automath* (AUTOmated MATHematics).
- Today, there is much interest in many approaches to proof checking for verification of computer hardware and software.
- Many theorem provers have been built to mechanically check mathematics and computer science reasoning (e.g. Isabelle, HOL, Coq, etc.).

The problem with formal logic

- No logical language has the criteria expected of a language of mathematics.
  - A logical language does not have *mathematico-linguistic* categories, is *not universal* to all mathematicians, and is *not a good communication medium*.
  - Logical languages make fixed choices (*first versus higher order, predicative versus impredicative, constructive versus classical, types or sets*, etc.). But different parts of mathematics need different choices and there is no universal agreement as to which is the best formalism.
  - A logician reformulates in logic their *formalization* of a mathematical-text as a formal, complete text which is structured considerably *unlike* the original, and is of little use to the *ordinary* mathematician.
  - Mathematicians do not want to use formal logic and have *for centuries* done mathematics without it.

- *So, mathematicians kept to* CML.

- We would like to find an alternative to CML which avoids some of the features of the logical languages which made them unattractive to mathematicians.

Full formalization difficulties: choices

A CML-text is structured differently from a fully formalized text proving the same facts. *Making the latter involves extensive knowledge and many choices:*

- The choice of the *underlying logical system.*
- The choice of *how concepts are implemented* (equational reasoning, equivalences and classes, partial functions, induction, etc.).
- The choice of the *formal foundation*: a type theory (dependent?), a set theory (ZF? FM?), a category theory? etc.
- The choice of the *proof checker*: Automath, Isabelle, Coq, PVS, Mizar, ...

An issue is that one must in general commit to one set of choices.

Full formalization difficulties: informality

Any informal reasoning in a CML-text will cause various problems when fully formalizing it:

- A single (big) step may need to expand into a (series of) syntactic proof expressions. Very long expressions can replace a clear CML-text.

- The entire CML-text may need *reformulation* in a fully *complete* syntactic formalism where every detail is spelled out. New details may need to be woven throughout the entire text. The text may need to be "turned inside out".

- Reasoning may be obscured by *proof tactics*, whose meaning is often *ad hoc* and implementation-dependent.

Regardless, ordinary mathematicians do not find the new text useful.

Coq example

draft documents	✗
public documents	✗
computations and proofs	✓

From Module `Arith.Plus` of Coq standard library (`http://coq.inria.fr/`).

```
Lemma plus_sym : (n,m:nat)(n+m)=(m+n).
Proof.
Intros n m ; Elim n ; Simpl_rew ; Auto with arith.
Intros y H ; Elim (plus_n_Sm m y) ; Simpl_rew ; Auto with arith.
Qed.
```

Where do we start? de Bruijn's Mathematical Vernacular MV

- *De Bruijn's Automath not just [...] as a technical system for verification of mathematical texts, it was rather a life style with its attitudes towards understanding, developing and teaching mathematics....The way mathematical material is to be presented to the system should correspond to the usual way we write mathematics. The only things to be added should be details that are usually omitted in standard mathematics.*

- MV is faithful to CML yet is formal and avoids ambiguities.

- MV is close to the usual way in which mathematicians write.

- MV has a syntax based on linguistic categories not on set/type theory.

- MV is weak as regards correctness: the rules of MV mostly concern *linguistic* correctness, its types are mostly linguistic so that the formal translation into MV is satisfactory *as a readable, well-organized text*.

Problems with MV

- MV makes many logical and mathematical choices which are best postponed.
- MV incorporates certain correctness requirements, there is for example a hierarchy of types corresponding with sets and subsets.
- MV is already *on its way* to a full formalization, while we want the option of remaining *closer to* a given informal mathematical content.
- We want a $formal$ language MathLang which •has the advantages of CML but not its disadvantages and •respects CML content.
- *MV does not respect* CML *content.*

What is the aim for MathLang?

Can we formalise a CML text, avoiding as much as possible the ambiguities of natural language, while still guaranteeing the following four goals?

1. The formalised text looks very much like the original CML text (and hence the content of the original CML text is respected).

2. The formalised text can be fully manipulated and searched in ways that respect its mathematical structure and meaning.

3. Steps can be made to do computation (via computer algebra systems) and proof checking (via proof checkers) on the formalised text.

4. This formalisation of text is not much harder for the ordinary mathematician than LaTeX. *Full formalization down to a foundation of mathematics is not required*, although allowing and supporting this is one goal.

(No theorem prover's language satisfies these goals.)

Starting point for MathLang: MV and WTT

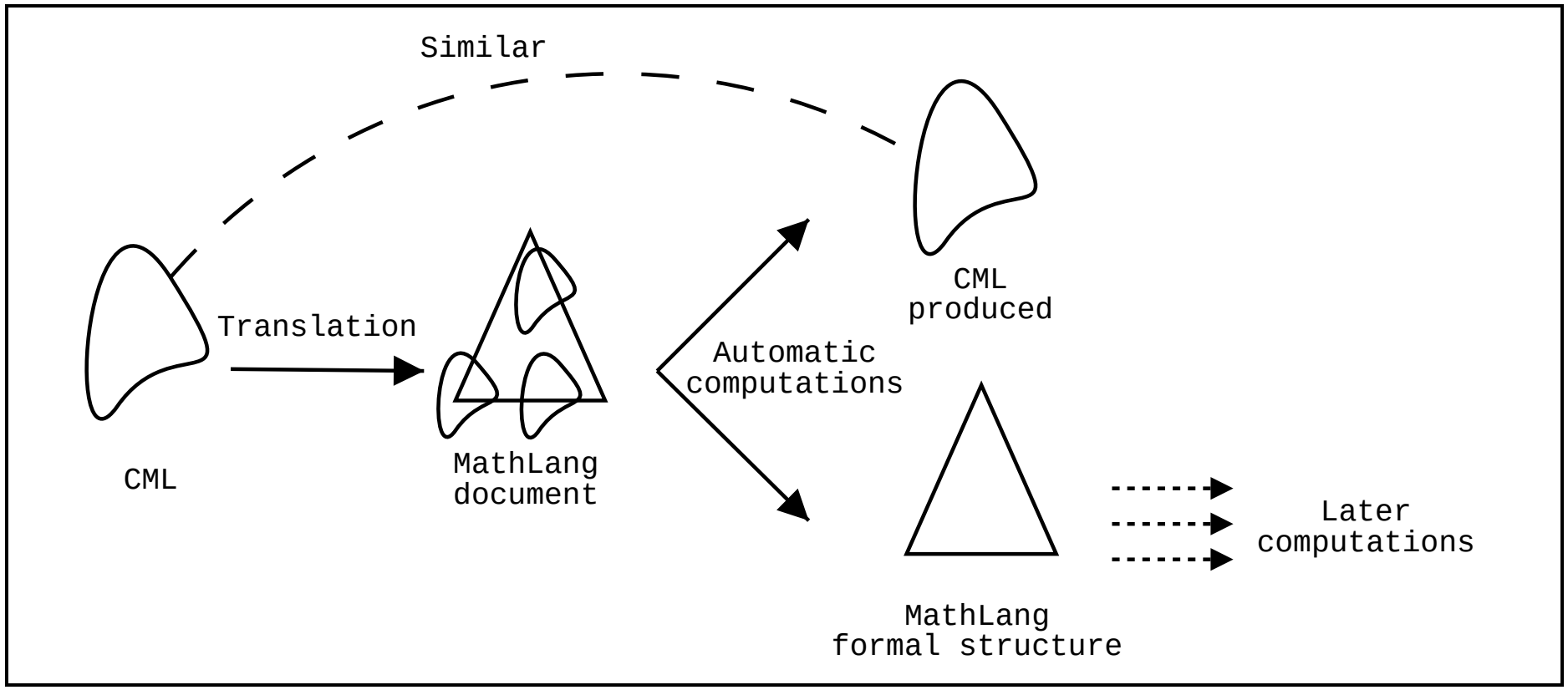
- MV was an initial inspiration for MathLang. But MV fails on goal 1.
- Weak Type Theory, WTT [2], is MV minus the added logic.
- Although in many ways WTT succeeds and improves on MV, it still fails on goal 1. A WTT text is not close to its CML original.
- With MathLang, we start from WTT, add some features, and investigate how to integrate it with natural language text.
- Our ongoing development of MathLang is driven by testing it in translating a set of sample texts chosen to cover a large portion of CML usages, both current and historical.

MathLang

draft documents	✓
public documents	✓
computations and proofs	✓

- A MathLang text captures the grammatical and reasoning aspects of mathematical structure for further computer manipulation.
- A *weak type system* checks MathLang documents at a grammatical level.
- A MathLang text remains *close* to its CML original, allowing confidence that the CML has been captured correctly.
- We have been developing ways to weave natural language text into MathLang.
- MathLang aims to eventually support *all encoding uses*.

Process of translation into MathLang



- The CML view of a MathLang text should match the mathematician's intentions.
- The formal structure should be suitable for various automated uses.

Linguistic categories in WTT and MathLang

- At the *atomic* level, WTT has separate syntactic categories for *variables*, *constants*, and *binders*. The latest MathLang uses one syntactic category and instead distinguishes these roles via weak types.

- At the *phrase* level, there are *terms*, *sets*, *nouns*, and *adjectives*. (Manuel's talk will give details on how this is handled in the latest MathLang.)

- At the *sentence* level, there are *statements* and *definitions*.

- At the *discourse* level, WTT has *contexts*, *lines*, *books*, and *prefaces*. The latest MathLang replaces these by *blocks* and *scoping* operators.

Generally, each syntactic category has a corresponding *weak type*.

Examples of linguistic categories

- Terms: the triangle ABC; the center of $\boxed{ABC}$; $d(\boxed{x}, \boxed{y})$.
- Nouns: a triangle; an edge of $\boxed{\text{ABC}}$; a group.
- Adjectives: equilateral $\boxed{\text{triangle}}$; prime $\boxed{\text{number}}$; Abelian $\boxed{\text{group}}$.
- Statements: $\boxed{P}$ lies between $\boxed{Q}$ and $\boxed{R}$; $\boxed{5} \geq \boxed{3}$; $\boxed{\text{AB}}$ is $\boxed{\text{an edge of ABC}}$.
- Definition: a number p is prime whenever $\boxed{\cdots}$.

MathLang example

Definition 1. A *Fermat-sum* is a natural number which is the sum of two squares of natural numbers.

Lemma 2. The product of a square and a Fermat-sum is a Fermat sum.

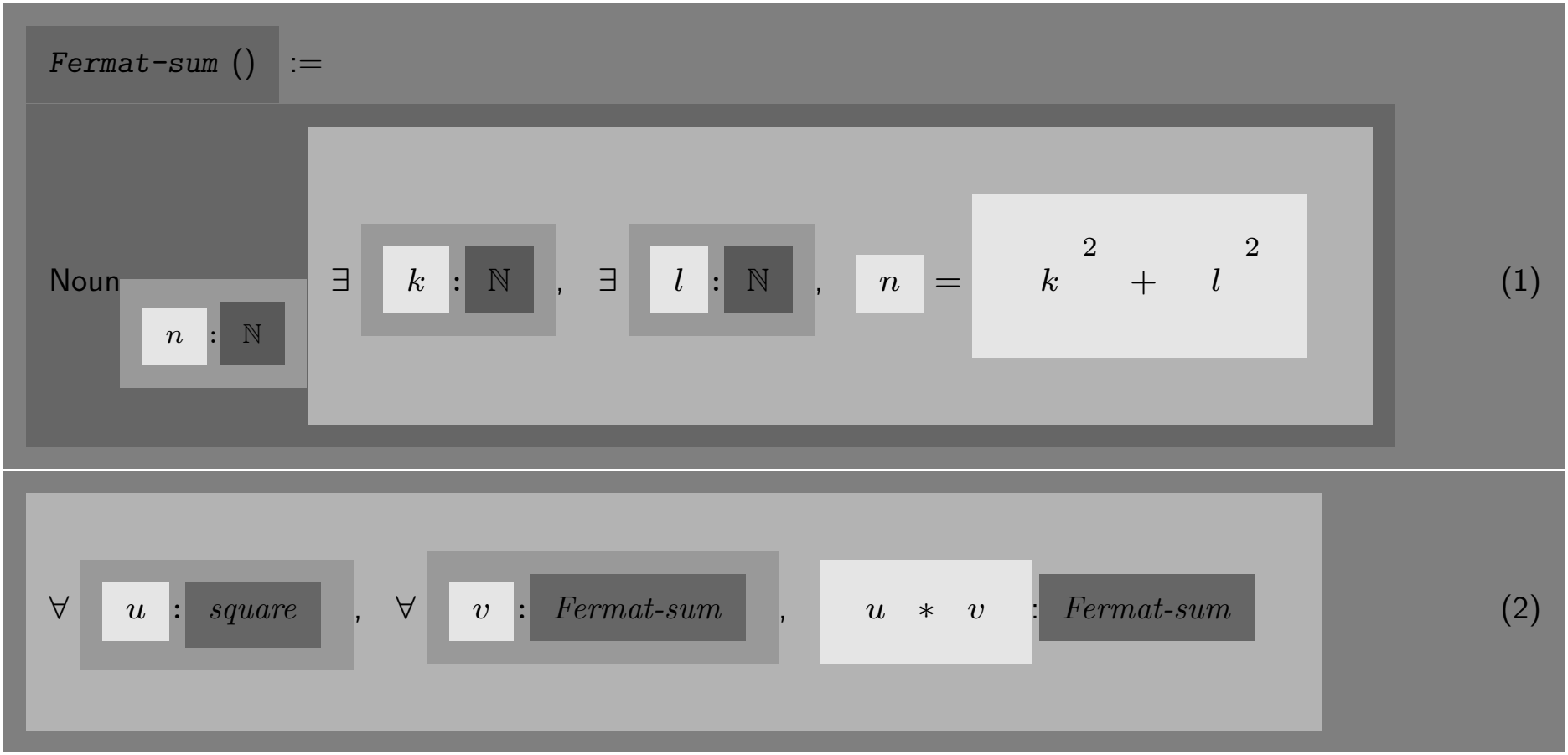
In an older MathLang version, the above text could be translated as the following two *lines*:

$$a\ \mathit{Fermat\text{-}sum}\ := \mathtt{Noun}_{n\in\mathbb{N}}\exists_{k\in\mathbb{N}}\exists_{l\in\mathbb{N}}(n = k^2 + l^2)$$

$$\forall_{u:\ a\ \mathit{square}}\forall_{v:\ a\ \mathit{Fermat\text{-}sum}}(uv :\ \ a\ \mathit{Fermat\text{-}sum})$$

We can also give the following interesting *views* of this example.

MathLang example: Symbolic structure view



MathLang example: CML view

Definition 3. [Fermat-sum]

A *Fermat-sum* is

a natural number which is the sum of two squares of natural numbers . 1

Lemma 4.

The product of a *square* and a *Fermat-sum* is a *Fermat-sum* . 2

Another MathLang example

T Terms | S Sets | N Nouns | P Statements | Z Declarations | Γ Context

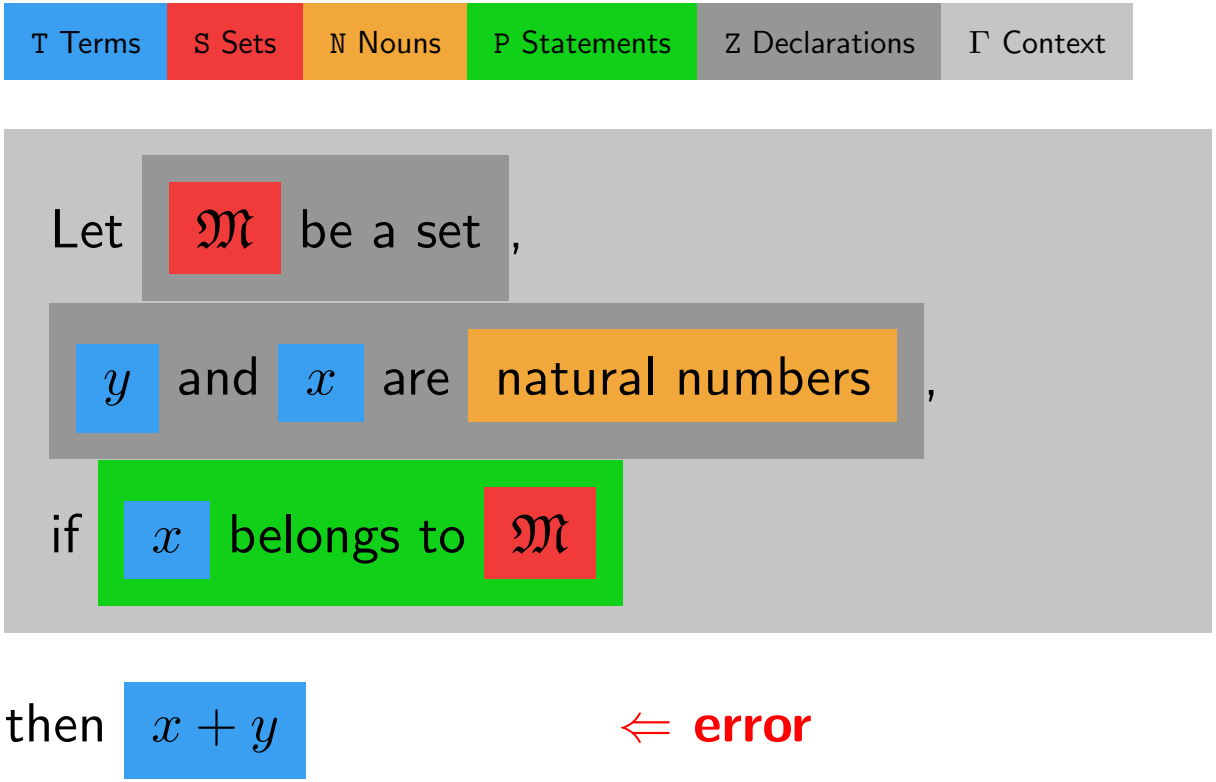
Let $\mathfrak{M}$ be a set ,

y and x are natural numbers ,

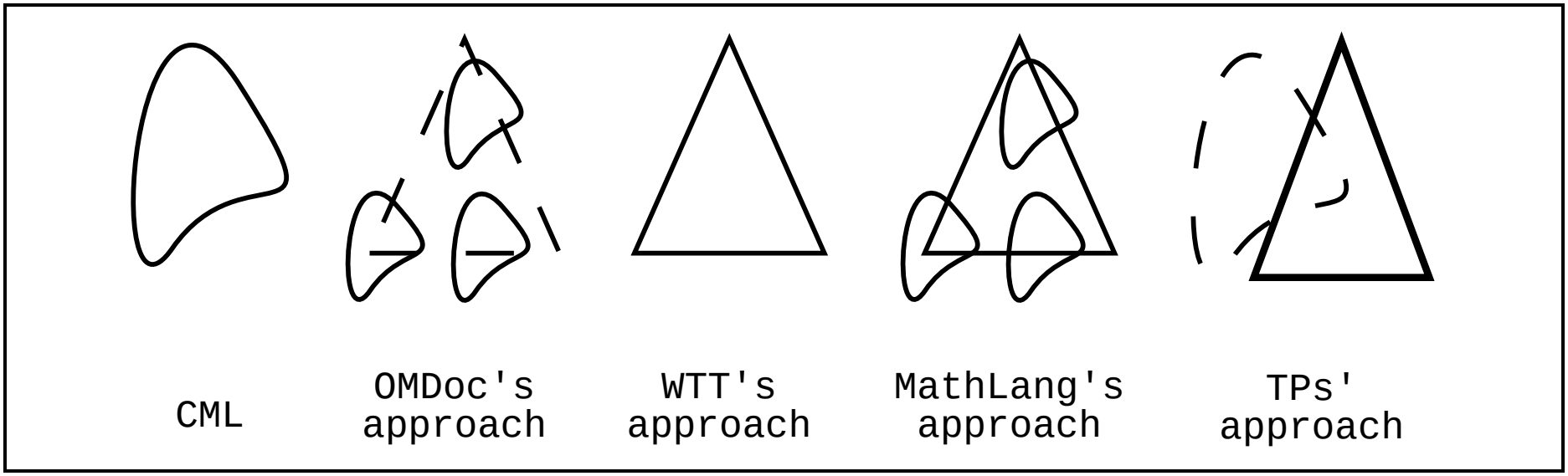
if x belongs to $\mathfrak{M}$

then $x + y = y + x$

Another MathLang example: Type checking



Various approaches to representing mathematics



We visually summarize the approaches. Blobs represent natural language text whose structure is not understood by the computer. A broken blob is text maintained separately, not as part of the data structure. Triangles represent a tree-shaped structures understood by the computer. Solid triangles represent additional computer-checked well-formedness conditions. A heavy solid triangle represents full formalization.

Additional comparison with other related work

- *Galina* serves as a *command language* for Coq, aimed at full formalization.
- The *mathematical vernacular* of *ΩMEGA* gives CML-like *views* of fully/partially formalized proofs.
- The basic languages of *Mizar* and Isar preserve the mathematical content. They are aimed at full formalization. Their syntax does not give the same expressive freedom to the mathematician as CML.
- In the *Theorema project* computer algebra systems, the provers are designed to imitate the proof style humans employ in their proving attempts. The proofs can be produced in human-readable style. However, this is done by *post-processing* a fully formal proof.
- The typed functional programming language *GF* can define languages such as fragments of natural languages, programming languages, and formal calculi. GF is based on Martin-Löf's type theory.

Some points to consider

- We do not at all assume/prefer one type/logical theory instead of another.
- The formalisation of a language of mathematics should separate the questions:
  - *which type/logical theory is necessary for which part of mathematics*
  - *which language should mathematics be written in*.
- Mathematicians don't usually know or work with type/logical theories.
- Mathematicians usually *do* mathematics (manipulations, calculations, etc), but are not interested in general in reasoning *about* mathematics.

Conclusion

- MathLang aims to support non-fully-formalized mathematics practiced by the ordinary mathematician as well as work toward full formalization.
- MathLang aims to handle mathematics as expressed in natural language as well as symbolic formulas.
- MathLang aims to do some amount of type checking even for non-fully-formalized mathematics. This corresponds roughly to grammatical conditions.
- MathLang aims for a formal representation of CML texts that closely corresponds to the CML conceived by the ordinary mathematician.
- MathLang aims to support automated processing of mathematical knowledge.
- MathLang aims to be independent of any foundation of mathematics.

References

[1] Kamareddine, F., L. Laan, and R. Nederpelt: 2002, 'Types in logic and mathematics before 1940'. *Bulletin of Symbolic Logic* **8**(2), 185–245.

[2] Kamareddine, F., and R. Nederpelt: 2004, A refinement of de Bruijn's formal language of mathematics. Journal of *Logic, Language and Information*. Kluwer Academic Publishers.

[3] Kamareddine, F., Maarek, M., and Wells, J.B.: 2004, MathLang: An experience driven language of mathematics, Electronic Notes in Theoretical Computer Science 93C, pages 123-145. Elsevier.

[4] F. Kamaredine, M Maarek, and J.B. Wells. Flexible encoding of mathematics on the computer. 2004.

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

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