Um Ceclo de Computeraçao

Fairouz Kamareddine

Sexta-Feira, 1 Octobre 2010

Brasiliá 2010

Welcome to the fastest developing and most influential subject: Computer Science

- Computer Science is by nature highly applied and needs much precision, foundation and theory.
- Computer Science is highly interdisciplinary bringing many subjects together in ways that were not possible before.
- Many recent scientific results (e.g., in chemistry) would not have been possible without computers.
- The Kepler Conjecture: no packing of congruent balls in Euclidean space has density greater than the density of the face-centered cubic packing.
- Sam Ferguson and Tom Hales proved the Kepler Conjecture in 1998, but it was not published until 2006.
- The Flyspeck project aims to give a formal proof of the Kepler Conjecture.

A Century of Complexity

	1900	2000
Main way information travels in society:	paper	electric signals, radio
Number of parts in complex machine:	10,000 (locomotive)	1,000,000,000 (CPU)
Worst consequences of single machine failure:	100s die	end of all life?
Likelihood a machine includes a computer:	very low	very high

We are playing God and creating new beings (the machines). These must function well or disasters will happen.

The Need for Some Kind of Formalism

- Because of the increasing interdependency of systems and the faster and more automatic travel of information, failures can have a wide impact. So *correctness* is important.

- Modern technological systems are just too complicated for humans to reason about unaided, so *automation* is needed.

- Systems have so many possible states that *testing* is often impractical. It seems that *proofs* are needed to cover infinitely many situations.

- So *some* kind of formalism is needed to *aid in design* and to *ensure safety*.

What Kind of Formalisms?

A reasoning formalism should *at least* be:

- *Correct*: Only correct statements can be "proven".
- *Adequate*: Needed properties in the problem domain can be stated and proved.
- *Feasible*: The resources (money, time) used in stating and proving needed properties must be within practical limits.

What Kind of Formalisms?

Assuming a minimally acceptable formalism, we would also like it to be:

- *Efficient*: Costs of both the reasoning process *and* the thing being reasoned about should be minimized.
- *Supportive of reuse*: Slight specification changes should not force reproving properties for an entire system. Libraries of pre-proved statements should be well supported.
- *Elegant*: The core of the reasoning formalism should be as simple as possible, to aid in reasoning about the formalism itself.

Logics, Types, and Rewriting are essential for Automation

Logics, types, and rewriting are

- elegant, as we can formulate and (automate) clear rules of how they work (e.g., from A and $A \to B$ we can deduce B),
- adequate (we can express a lot in these tiny formalisms), and
- able to be shown correct.

Logics, types, and rewriting have existed in various since from the times of the ancient Babylonians and Greeks (e.g., Euclid, Aristotle, etc.).

The need for Automation is old but there was no computers

- Logic is *OLD*. Mathematics is *OLD*. But, *SO IS* computer science.
- Assume a problem Π,
  - If you *give* me an algorithm to solve Π, I can check whether this algorithm really solves Π.
  - But, if you ask me to *find* an algorithm to solve Π, I may go on forever trying but without success.
- But, this result was already found by Aristotle: Assume a proposition Φ.
  - If you *give* me a proof of Φ, I can check whether this proof really proves Φ.
  - But, if you ask me to *find* a proof of Φ, I may go on forever trying but without success.
- In fact, *programs* are *proofs* and much of computer science in the early part of the 20th century was created by the need to automate.

Proofs? Logics? What are they?

- A proof is the *guarantee* of some statement provided by a rigorous *explanation* stated using some *logic*.
- A logic is a formalism for statements and proofs of statements. A logic usually has *axioms* (statements "for free") and *rules* for combining already proven statements to prove more statements.
- Why do we believe the explanation of a proof? Because a proved statement is derived step by step from explicit assumptions using a trusted logic.
- There has been an explosion of new logics in the 20th century. How do we know which ones to trust? Fund us and we will tell you ...

A Brief History of Logic (Aristotle)

- Aristotle (384–322 B.C.) wanted a set of rules that would be powerful enough for most intuitively valid proofs.

- Aristotle correctly stated that *proof search* is harder than *proof checking*:

  Given a proof of a statement, one can check that it is a correct proof.
  Given a statement, one may not be able to find the proof.

  Aristotle's intuitions on this have been confirmed by Gödel, Turing, and others.

A Brief History of Logic (Leibniz)

- Leibniz (1646–1717) conceived of *automated deduction*, i.e., to find
  - a language L in which arbitrary concepts could be formulated, and
  - a machine to determine the correctness of statements in L.

- Such a machine can not work for every statement according to Aristotle and (later results by) Gödel and Turing.

A Brief History of Logic (Cantor, Peano, Frege)

The late 1800s saw the beginnings of serious formalization:

- Cantor began formalizing set theory [2, 3] and made contributions to number theory.
- Peano formalized arithmetic [31], but did not treat logic or quantification.
- Frege's *Begriffsschrift* [12] (1879) was the first formalisation of logic which presented logical concepts via symbols rather than natural language. Frege's *Grundgesetze der Arithmetik* [14, 18], called later by others Naive Set Theory (NST), could handle elementary arithmetic, set theory, logic, and quantification.

A Brief History of Logic (Frege's Set Theory)

- Frege's NST allowed a precise definition of the vital concept of the *function*. As a result, NST could include not only functions that take numbers as arguments and return numbers as results, but also functions that can take and return other sorts of arguments, *including functions*. These powerful functions were the key to the formalization of logic in NST.

- Frege was cautious: ordinary functions could only take "objects" as arguments, not other functions. However, to gain important expressive power, he allowed a way to turn a function into an object representing its graph.

- Unfortunately, this led to a *paradox*, due to the implicit possibility of *self-application* of functions.

A Brief History of Logic (Russell's Paradox)

- In 1902, Russell suggested [35] and Frege completed the argument [17] that a *paradox* could occur in NST. First, one can define S to be "the set of all sets which do not contain themselves". Then, one can prove *both* of these statements in NST:

$$S \in S \qquad\qquad S \notin S$$

- In fact, the same paradox could be encoded in the systems of Cantor and Peano. As a result, all three systems were *inconsistent* — not only could every true statement be proved but also every false one! (Three-valued logic can solve this, but is unsatisfactory for other reasons.) Logic was in a *crisis*.

- In 1908, Russell suggested the use of *types* to solve the problem [37].

A Brief History of Types (Euclid)

- Euclid's *Elements* (circa 325 B.C.) begins with:

  1. A *point* is that which has no part;
  2. A *line* is breadthless length.

     ⋮

  15. A *circle* is a plane figure contained by one line such that all the straight lines falling upon it from one point among those lying within the figure are equal to one another.

- Although the above seems to merely *define* points, lines, and circles, it shows more importantly that Euclid *distinguished* between them. Euclid always mentioned to which *class* (points, lines, etc.) an object belonged.

A Brief History of Types (Euclid)

- By distinguishing classes of objects, Euclid prevented undesired situations, like considering whether two points (instead of two lines) are parallel.

- *Undesired* results? Euclid himself would probably have said: *impossible* results. When considering whether two objects were parallel, intuition implicitly forced him to think about the *type* of the objects. As intuition does not support the notion of parallel points, he did not even *try* to undertake such a construction.

- In this manner, types have always been present in mathematics, although they were not noticed explicitly until the late 1800s. If you have studied geometry, then you have some (implicit) understanding of types.

A Brief History of Types (Paradox Threats)

- Starting in the 1800s, mathematical systems became less intuitive, for several reasons:
  - Very complex or abstract systems.
  - Formal systems.
  - Something with less intuition than a human using the systems: a computer.

- These situations are *paradox threats*. An example is Frege's NST. In such cases, there is not enough intuition to activate the (implicit) type theory to warn against an impossible situation. Reasoning proceeds within the impossible situation and then obtains a result that may be wrong or paradoxical.

Example Failures due to Type Errors

An untyped computer program may receive instructions from a first-year student to add the number 3 to the word "`four`" (instead of the number 4). The computer is unaware that "`four`" is not a number and the result of $3 +$ "`four`" is unpredictable. The computer may

- give an answer that is clearly wrong (for example, `true`),
- give no answer at all, or
- give an answer that is not so clearly wrong (for example, 6).

Especially the last situation is highly undesirable.

A Brief History of Types (Russell)

- To avoid the paradoxes of the systems of Cantor, Peano, and Frege, Russell prescribed avoiding self-reference and self-application in his "vicious circle principle":

  Whatever involves all *of a collection must not be one of the collection.*

- Russell implemented this in his Ramified Theory of Types (RTT) [37] which used *types* and *orders*. Self-application was prevented by forcing functions of order k to be applied only to arguments of order less than k.

- This was carried out further by Russell and Whitehead in the famous *Principia Mathematica* [39] (1910-1912), which founded mathematics on logic, as far as possible, avoiding paradoxes.

A Brief History of Types (Russell)

- For example, in RTT, one can define a function "+" which is restricted to be applied only to integers.

- Although RTT was correct, unlike NST, the types of RTT have turned out instead to be *too restrictive* for mathematics and computer science where fixed points (to mention one example) play an important role. RTT also forces duplication of the definitions of the number system, the boolean algebra, etc., at *every* level.

- The exploration of the middle ground between these two extremes has led to many systems, most of them in the context of the λ-calculus, the first higher-order *rewriting* system.

A Quick Introduction to Rewriting

We all know how to do *algebra*:

$$
\begin{array}{rll}
 & (a+b)-a & \text{by rule } x+y=y+x \\
= & (b+a)-a & \text{by rule } x-y=x+(-y) \\
= & (b+a)+(-a) & \text{by rule } (x+y)+z=x+(y+z) \\
= & b+(a+(-a)) & \text{by rule } x+(-x)=0 \\
= & b+0 & \text{by rule } x+0=x \\
= & b &
\end{array}
$$

Rewriting is the action of replacing a subexpression which is matched by an instance of one side of a rule by the corresponding instance of the other side of the same rule. If you have studied algebra, then you are skilled at rewriting.

Important Issues in Rewriting

- *Orientation*: Usually, most rules can only be used from left to right as in $x + 0 \rightarrow x$. Forward use of the oriented rules represents progress in computation. Unoriented rules usually do trivial work as in $x + y = y + x$.

- *Termination*: It is desirable to show that rewriting halts, i.e., to avoid infinite sequences of the form $P \rightarrow P_1 \rightarrow P_2 \rightarrow \cdots$.

- *Confluence*: The result of rewriting is independent of the order in the rules are used. For example, $1 + 2 + 3$ should rewrite to 6, no matter how we evaluate it.

A Brief History of Rewriting (Ancients)

- When the Greeks introduced logic they did not have modern-style rewriting.
- The Babylonians on the other hand, developed techniques for symbolic computations through their work on algebra. This can be viewed as rewriting.
- The Arabs of course first introduced algebra in close to its modern form.

A Brief History of Rewriting (λ-Calculus)

- In the late 1800s, Frege identified the *abstraction principle*: Any expression mentioning some symbol in zero or more places can be turned into a function by abstracting over that symbol.

- Introduced in the 1930s, Church's λ-calculus made function abstraction an *operator*. For example, $(\lambda x.\, x + 5)$ represents the (unnamed) mathematical function which takes as input any number and returns as output the result of adding 5 to that number.

- The λ-calculus provides *higher-order* rewriting, allowing equations like:

$$f(\underline{(\lambda x.\, x + (1/x))5}) = f(5 + \underline{(1/5)}) = f(\underline{5 + 0.2}) = f(5.2)$$

A Brief History of Rewriting (λ-Calculus)

- The type-free λ-calculus, which can be seen as a small computer programming language, is an excellent theory of functions — it can represent all computable functions.

- Church intended the *type-free* λ-calculus with logical operators to provide a foundation for mathematics. Unfortunately, Russell's paradox could also be encoded in the type-free λ-calculus, rendering its use for logic incorrect.

- Church introduced the simply typed λ-calculus (STLC) [7] to provide logic while avoiding Russell's paradox in a manner similar to RTT. Unfortunately, like RTT, the STLC is too restrictive. A modern, slightly less restrictive descendant of this approach is the so-called "higher-order logic" (HOL).

The Convergence of Logics, Types, and Rewriting

- Heyting [20], Kolmogorov [28], Curry and Feys [8] (improved by Howard [22]), and de Bruijn [30] all observed the "*propositions as types*" or "*proofs as terms*" (PAT) correspondence.

- In PAT, logical operators are embedded in the types of λ-terms rather than in the terms and λ-terms are viewed as proofs of the propositions represented by their types.

- Advantages of PAT include the ability to manipulate proofs, easier support for independent proof checking, the possibility of the extraction of computer programs from proofs, and the ability to prove the consistency of the logic via the termination of the rewriting system.

An example of a computable function/solvable problem

- E.g., 1.5 chicken lay down 1.5 eggs in 1.5 days.
- How many eggs does 1 chicken lay in 1 day?
- 1.5 chicken lay 1.5 eggs in 1.5 days.
- Hence, 1 chicken lay 1 egg in 1.5 days.
- Hence, 1 chicken lay 2/3 egg in 1 day.

Unsolvability of the Barber problem

- *which man barber in the village shaves all and only those men who do not shave themselves?*
- If John was the barber then
  - *John shaves Bill* $\Longleftrightarrow$ *Bill does not shave Bill*
  - *John shaves x* $\Longleftrightarrow$ *x does not shave x*
  - *John shaves John* $\Longleftrightarrow$ *John does not shave John*
- *Contradiction.*

Unsolvability of the Russell set problem

- Another unsolvable problem: Give me the Russell set $R = \{x \mid x \notin x\}$.
- If R existed then $x \in R$ *iff* $x \notin x$.
  - If $R \in R$ then
    $(x \notin x)[x := R]$ and so $R \notin R$.
    Contradiction.
  - If $R \notin R$ then
    $(x \notin x)[x := R]$ and so $R \in R$.
    Contradiction.
- What about the problem:
- *Find an algorithm which takes any program P and input x and tells you whether P halts or loops with input x.*

- Aristotle already knew that for a proposition Φ.
  - If you *give* me a proof of Φ, I can check whether this proof really proves Φ.
  - But, if you ask me to *find* a proof of Φ, I may go on forever trying but without success.

- Aristotle used logic to reason about everything (law, farming, medicine,...)

- Euclides set the grounds for a deductive method for reasoning about geometry (approaching the dream of a machine that can do some part of our work).

- At the glorious times of the palaces in Alhambra and Andalucia, princes used to study mathematics for pleasure. Their courts had musicians, poets, math teachers, etc. These teachers already insisted that Maths must be taught and developed using logic. This is one of the main themes of the research of Frege and Russell.

- In the 17th century, Leibniz wanted to use logic to prove the existence of God.

Why did computer science kick off in the 20th century?

In the 19th century, the *need for a more precise* style in mathematics arose, *because controversial results* had appeared in *analysis*.

- 1821: Many of these controversies were solved by the work of Cauchy. E.g., he introduced *a precise definition of convergence* in his *Cours d'Analyse* [4].

- 1872: Due to the more *exact definition of real numbers* given by Dedekind [10], the rules for reasoning with real numbers became even more precise.

- 1895-1897: Cantor began formalizing *set theory* [2, 3] and made contributions to *number theory*.

Formal systems in the 19th century

- 1889: *Peano* formalized *arithmetic* [31], but did not treat logic or quantification.

- 1879: *Frege* was not satisfied with the use of *natural language in mathematics*:

  > ". . . I found *the inadequacy of language to be an obstacle*; no matter how unwieldy the expressions I was ready to accept, I was less and less able, as the relations became more and more complex, to attain the precision that my purpose required."
  >
  > (*Begriffsschrift*, Preface)

  Frege therefore presented *Begriffsschrift* [12], the first formalisation of logic giving logical concepts via symbols rather than natural language.

Formal systems in the 19th century

> "[Begriffsschrift's] first purpose is to *provide us with the most reliable test of the validity of a chain of inferences* and to point out *every presupposition* that tries to sneak in unnoticed, so that its origin *can be investigated*."
>
> (*Begriffsschrift*, Preface)

- 1892-1903 Frege's *Grundgesetze der Arithmetik* [14, 18], could handle elementary arithmetic, set theory, logic, and quantification.
- Also in 1900, Hilbert, posed a list of problems at a conference in Paris.
- One very important question was: Can any logical statement have a proof or be disproved.
- More than 30 years later, this question was negatively answered by Turing (Turing machines), Goedel (incompleteness results) and Church (λ-calculus).

Can we solve/compute everything?

- Turing answered the question in terms of a *computer*. Turing's machines are so powerful: *anything that can ever be computed even on the most powerful computers, can also be computed on a Turing machine.*

- Church invented the λ-calculus, a *language for programming*. λ-calculus is so powerful: *anything that can ever be computed can be described in the λ-calculus.*

- Goedel's result meant that no absolute guarantee can be given that many significant branches of mathematics are entirely free of contradictions.

- This meant that: we can compute a very small (countable) amount compared to what we will never be able to compute (uncountable).

- Hilbert's dream was shattered. According to the great historian of Mathematics Ivor Grattan-Guinness, Hilbert behaved coldly towards Goedel.

How did Logic and mathematics influence programming languages?

- Untyped λ-calculus was adopted in LISP.
- Simply typed λ-calculus was adopted in theorem provers like HOL and was used to make sense of other programming languages (e.g., pascal).
- Then, simple types were extended to *polymorphic* (and other) types.
- These are used in programming languages like ML.
- And the search continues for better and better programming languages.
- *Types* continue to play an influential role in the design and implementation of programming languages.

Above all, Programs (or algorithms) are computable functions

The introduction of a *very general definition of function* was the key to the formalisation of logic. Frege defined the Abstraction Principle.

Abstraction Principle 1.

> *"If in an expression, [. . .] a simple or a compound* sign has one or more occurrences *and if we regard that sign as* replaceable *in all or some of these occurrences* by something else *(but everywhere by the same thing), then we call* the part that remains invariant in the expression *a* function, *and* the replaceable part *the* argument *of the function."*
>
> *(Begriffsschrift,* Section 9*)*

So, computers and programming languages have been invented, but how/what do we computerise/automate?

- Remember that we can only compute a small infinite number of functions (the size of the natural numbers). The rest is a huge infinite amount (the size of the reals) that we cannot compute.

- Even the small number of computable functions is not easy to compute (some computable functions are very hard and complex to compute that they might as well be noncomputable).

- But, we don't give up on the hard computable functions. We keep trying.

- Also, we don't avoid completely functions that are non computable. We try to compute useful approximations of the function. We still get a lot out by doing so.

- Also, we keep improving languages and tools of computation. The languages of programming still need a lot of improvements.

Some interesting areas of research at UNB

- Vision, Graphics, image processing: Bruno, camilo, Dibio, Flavio Vidal, Mylene, Ricardo de Queiroz,
- Parallel and distributed computation, networking: Alba, Aletia, Carla Castanho, Jacir,
- Bioinformatics: Alba, Maristela, Maria Emilia, Wilson
- Neural networks and mobile computing: Pedro Berger, Pricila,
- Music and computation: Aluizio, Marcio,
- Artificial intelligence and Robotics, Machine learning, air traffic control: Carla Koike, Flavio Vidal, Germana, Li, Marcelo, Marcus,

- Information systems and security: Celia, Jan, Jaoa, Jorge, Marcelo, Pedro Rezende,

- Theory of computation, Logics, types, rewriting and automation: Claudia, Flavio de Moura, Mauricio, Jose Ralha, Pedro Rezende

- Hardware verification: Ricardo Jacobi,

- Software engineering, web semantics, system architecture: Vander, Fernanda, Fernando, Genaina,

- Computers and Education: Homero, Marcio, Marco, Fatima,

- Computational linguistic: Jose ralha,

an example of how difficult computerisation is

- Nowadays, *computerization* is an essential feature of any field.
- What is the influence of computerization on the study of language.
- Which language? Arabic, English, French, German, Portuguese, ...
- Euclid's book on geometry was written in Greek in Alexandria and translated into many languages.
- The impressive library of Alexandria at that time was destroyed later.
- Attempts at recreating this library *electronically* are under way
- We need the computerization of a huge number of texts and books.
- Why computerize books? How do we computerize books? What problems do we encounter?

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 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.
```

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.)

MathLang example: CML view

Definition 2. [Fermat-sum]

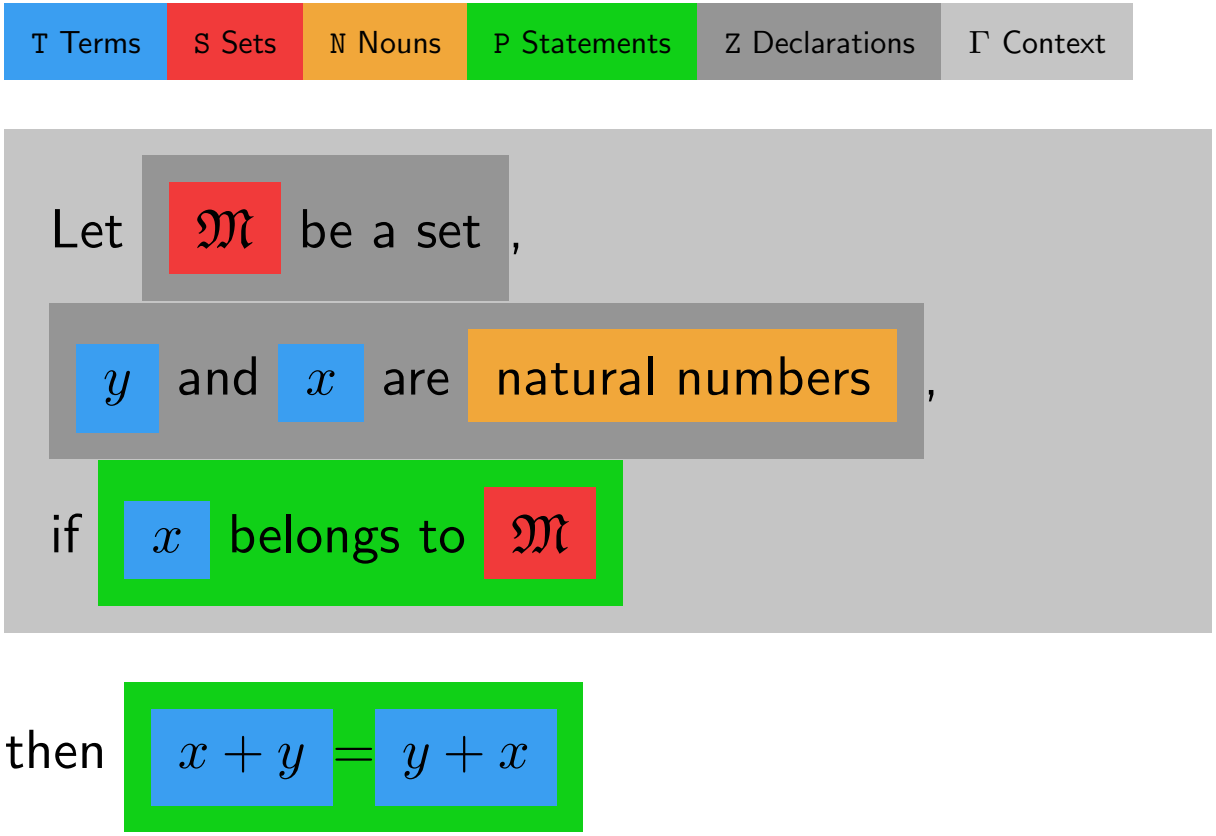
A *Fermat-sum* is

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

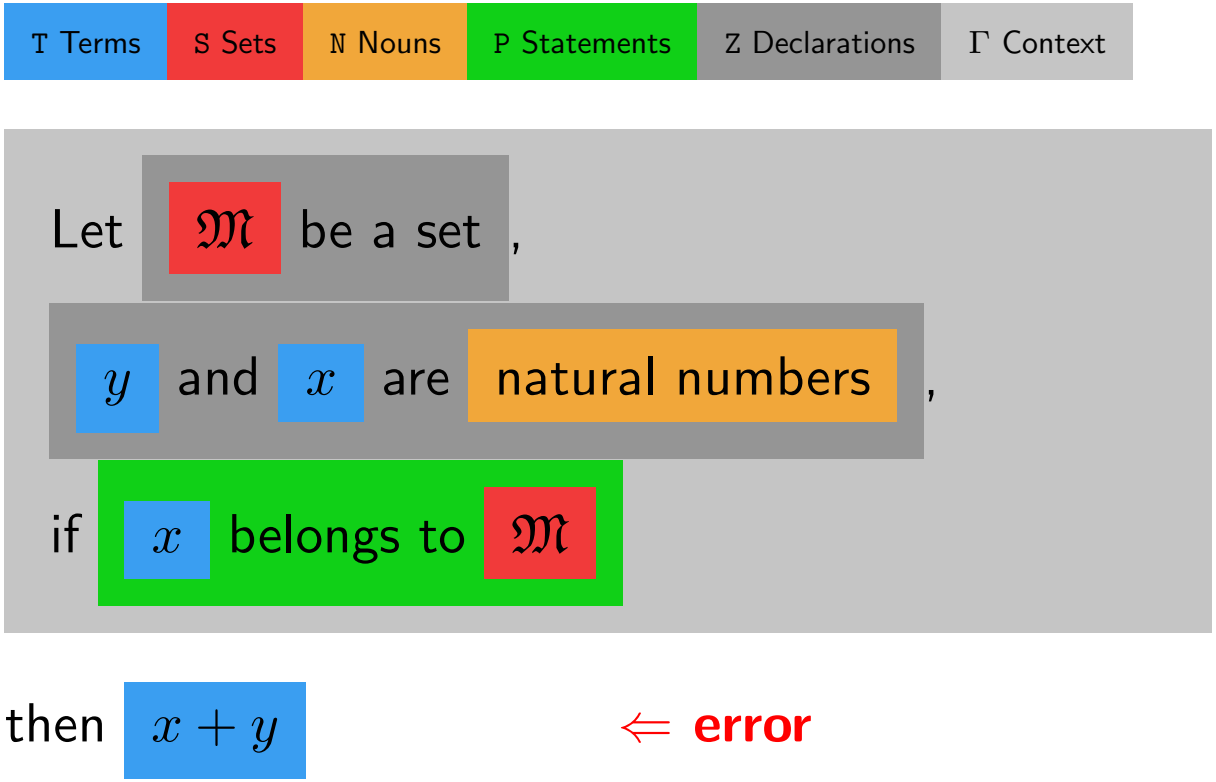
Lemma 3.

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

Another MathLang example



Another MathLang example: Type checking



References

[1] C. Burali-Forti. Una questione sui numeri transfiniti. *Rendiconti del Circolo Matematico di Palermo*, 11:154–164, 1897. English translation in [19], pages 104–112.

[2] G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (Erster Artikel). *Mathematische Annalen*, 46:481–512, 1895.

[3] G. Cantor. Beiträge zur Begründung der transfiniten Mengenlehre (Zweiter Artikel). *Mathematische Annalen*, 49:207–246, 1897.

[4] A.-L. Cauchy. *Cours d'Analyse de l'Ecole Royale Polytechnique*. Debure, Paris, 1821. Also as *Œuvres Complètes* (2), volume III, Gauthier-Villars, Paris, 1897.

[5] A. Church. A set of postulates for the foundation of logic (1). *Annals of Mathematics*, 33:346–366, 1932.

[6] A. Church. A set of postulates for the foundation of logic (2). *Annals of Mathematics*, 34:839–864, 1933.

[7] A. Church. A formulation of the simple theory of types. *The Journal of Symbolic Logic*, 5:56–68, 1940.

[8] H. B. Curry and R. Feys. *Combinatory Logic I*. Studies in Logic and the Foundations of Mathematics. North-Holland, Amsterdam, 1958.

[9] D.T. van Daalen. *The Language Theory of Automath*. PhD thesis, Eindhoven University of Technology, 1980.

[10] R. Dedekind. *Stetigkeit und irrationale Zahlen*. Vieweg & Sohn, Braunschweig, 1872.

[11] D.R. Dowty. *Introduction to Montague Semantics*. Kluwer Academic Publishers, 1980.

[12] G. Frege. *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*. Nebert, Halle, 1879. Also in [19], pages 1–82.

[13] G. Frege. *Grundlagen der Arithmetik, eine logisch-mathematische Untersuchung über den Begriff der Zahl*. , Breslau, 1884.

[14] G. Frege. *Grundgesetze der Arithmetik, begriffschriftlich abgeleitet*, volume I. Pohle, Jena, 1892. Reprinted 1962 (Olms, Hildesheim).

[15] G. Frege. Über Sinn und Bedeutung. *Zeitschrift für Philosophie und philosophische Kritik,* new series, 100:25–50, 1892.

[16] G. Frege. Ueber die Begriffschrift des Herrn Peano und meine eigene. *Berichte über die Verhandlungen der Königlich Sächsischen Gesellschaft der Wissenschaften zu Leipzig, Mathematisch-physikalische Klasse 48*, pages 361–378, 1896.

[17] G. Frege. Letter to Russell. English translation in [19], pages 127–128, 1902.

[18] G. Frege. *Grundgesetze der Arithmetik, begriffschriftlich abgeleitet*, volume II. Pohle, Jena, 1903. Reprinted 1962 (Olms, Hildesheim).

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

[19] Heijenoort, J. v. (ed.): 1967, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879–1931*. Cambridge, Massachusetts: Harvard University Press.

[20] A. Heyting. *Mathematische Grundlagenforschung. Intuitionismus. Beweistheorie.* Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer-Verlag, Berlin, 1934.

[21] D. Hilbert and W. Ackermann. *Grundzüge der Theoretischen Logik*. Die Grundlehren der Mathematischen Wissenschaften in Einzeldarstellungen, Band XXVII. Springer Verlag, Berlin, first edition, 1928.

[22] W. A. Howard. The formulaes-as-types notion of construction. In J. R[oger] Hindley and J[onathan] P. Seldin, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 479–490. Academic Press, 1980. An earlier version was privately circulated in 1969.

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

[24] 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.

[25] 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.

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

[27] F. Kamareddine, T. Laan, and R. Nederpelt. Revisiting the notion of function. *Logic and Algebraic programming*, 54:65–107, 2003.

[28] A. N. Kolmogorov. Zur Deutung der Intuitionistischen Logik. *Mathematisches Zeitschrift*, 35:58–65, 1932.

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

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

[29] MacLane, S.: 1972, *Categories for the Working Mathematician*. Springer.

[30] Rob Nederpelt, J. H. Geuvers, and Roel C. de Vrijer. *Selected Papers on Automath*, volume 133 of *Studies in Logic and the Foundations of Mathematics*. North-Holland, Amsterdam, 1994.

[31] G. Peano. *Arithmetices principia, nova methodo exposita*. Bocca, Turin, 1889. English translation in [19], pages 83–97.

[32] G. Peano. *Formulaire de Mathématique*. Bocca, Turin, 1894–1908. 5 successive versions; the final edition issued as *Formulario Mathematico*.

[33] F.P. Ramsey. The foundations of mathematics. *Proceedings of the London Mathematical Society,* 2nd series, 25:338–384, 1926.

[34] J.B. Rosser. Highlights of the history of the lambda-calculus. *Annals of the History of Computing*, 6(4):337–349, 1984.

[35] B. Russell. Letter to Frege. English translation in [19], pages 124–125, 1902.

[36] B. Russell. *The Principles of Mathematics*. Allen & Unwin, London, 1903.

[37] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908. Also in [19], pages 150–182.

[38] M. Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924. Also in [19], pages 355–366.

[39] Whitehead, A. and B. Russell: 1910^1, 1927^2, *Principia Mathematica*, Vol. I, II, III. Cambridge University Press.

[40] Zermelo, E.: 1908, 'Untersuchungen über die Grundlagen der Mengenlehre'. *Math. Annalen* **65**, 261–281.