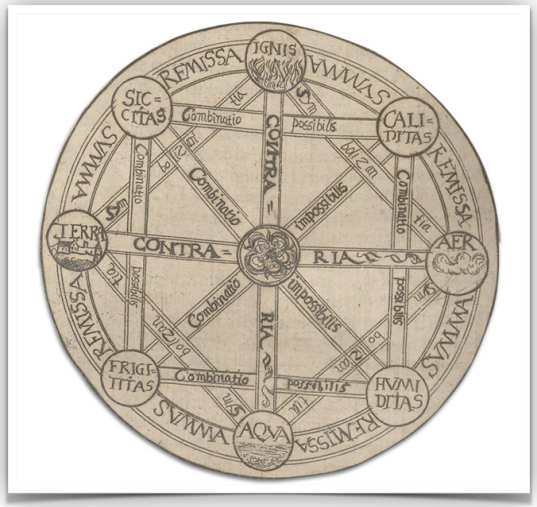
Hello everyone. It is a great honor for me to have the opportunity to address such a wide audience today. I would like to tell you about formalized mathematics, which is mathematics written in a formal language and verified with computers. I believe that one day formalized mathematics will change how mathematics is done, and that day may be nearer than we think.

By the way, you can download my slides with speaker notes and links to references at math.andrej.com.

What is formalized mathematics?

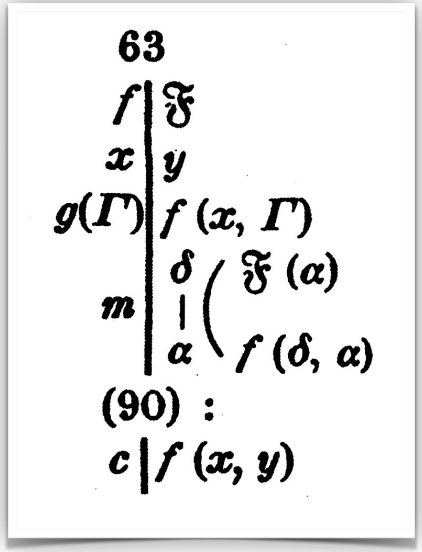
Let me first explain what precisely I mean by “formalized mathematics”.



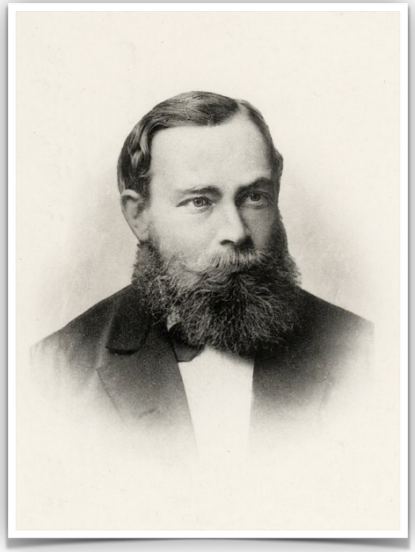
Characteristica universalis

Gottfried Leibniz

Already in the 17th century Gottfried Leibniz spoke of “characteristica universalis”, an imagined precise language and diagrammatical system able to express mathematical, scientific, and metaphysical concepts. His ideas spared much interest and never died out.

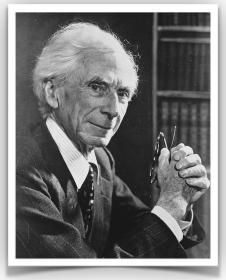


Begriffschrift

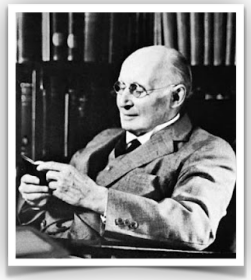


Gottlob Frege

About two centuries later mathematicians made serious progress in devising formal languages that could in fact express mathematical constructions and proofs. An important advance was made by Gottlob Frege who devised “Begriffschrift”, a diagrammatic system that was entirely formal and precise.



Bertrand Russell



Alfred North Whitehead

$*54{\cdot}43.\quad \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$

Dem.

$\vdash . *54{\cdot}26 . \supset \vdash :. \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$

$[*51{\cdot}231] \qquad \equiv . \iota`x \cap \iota`y = \Lambda .$

$[*13{\cdot}12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$

$\vdash . (1) . *11{\cdot}11{\cdot}35 . \supset$

$\vdash :. (\exists x, y) . \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . \alpha \cap \beta = \Lambda \qquad (2)$

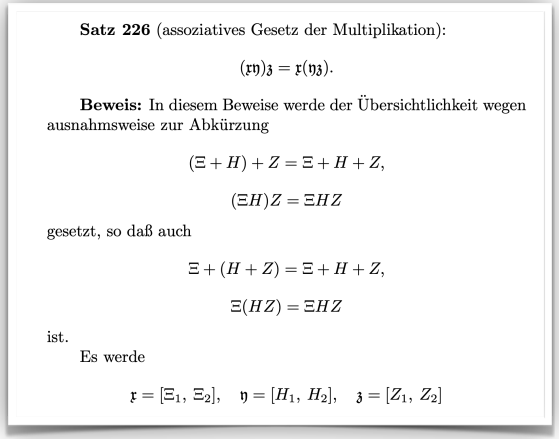
$\vdash . (2) . *11{\cdot}54 . *52{\cdot}1 . \supset \vdash . \text{Prop}$

From this proposition it will follow, when arithmetical addition has been defined, that $1 + 1 = 2$.

Principia Mathematica

Then at the beginning of the 20th century, in their monumental work “Principia Mathematica”, Bertrand Russell and Alfred North Whitehead, sought to reduce all of mathematics to pure logical principles and symbolism. The work inspired the development of formal logic in the 20th century.

It was claimed or expected that various formalism, such as first-order logic and Zermelo-Fraenkel set theory, could in principle be used to express all of mathematics with complete precision. Moreover, the formal expression could be checked for correctness mechanically. But could it realistically be done? Somebody had to try.



Satz 226 (assoziatives Gesetz der Multiplikation):

$$(\mathfrak{x}\mathfrak{y})\mathfrak{z} = \mathfrak{x}(\mathfrak{y}\mathfrak{z}).$$

Beweis: In diesem Beweise werde der Übersichtlichkeit wegen ausnahmsweise zur Abkürzung

$$(\Xi + H) + Z = \Xi + H + Z,$$

$$(\Xi H)Z = \Xi HZ$$

gesetzt, so daß auch

$$\Xi + (H + Z) = \Xi + H + Z,$$

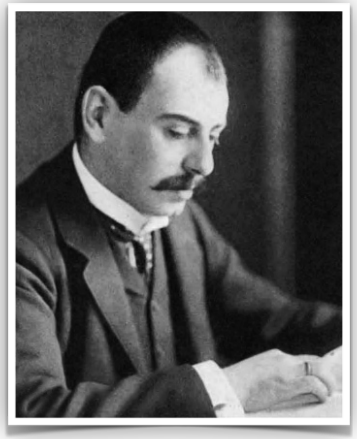
$$\Xi(HZ) = \Xi HZ$$

ist.

Es werde

$$\mathfrak{x} = [\Xi_1, \Xi_2], \quad \mathfrak{y} = [H_1, H_2], \quad \mathfrak{z} = [Z_1, Z_2]$$

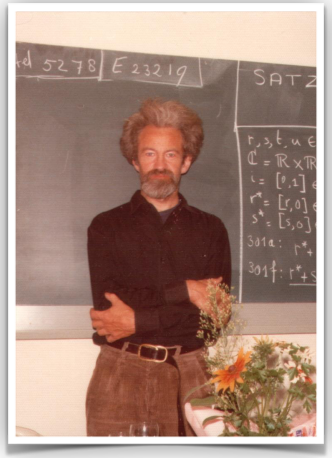
Gründlagen der Analysis



Edmund Landau

When I was a student, I heard that someone had formalized and checked with computer an entire book of analysis. The book was Edmund Landau's "Gründlagen der Analysis".

My teachers implied that this was a heroic but slightly insane task which only demonstrated what everyone knew already, namely that mathematics could indeed be formalized. And now that the point had been made, we could all proceed with business as usual and forget about formalizing mathematics.



Bert van Benthem Jutting

```
z@satz226:=t13".3226":is(ts(ts(x,y),z),ts(x,ts(y,z)))

assts1:=satz226:is(ts(ts(x,y),z),ts(x,ts(y,z)))
assts2:=symis(cx,ts(ts(x,y),z),ts(x,ts(y,z)),assts1):is
(ts(x,ts(y,z)),ts(ts(x,y),z)) +3227
c@t1:=tr3is(real,pl"r"(pl"r"(a,b),c),pl"r"(a,pl"r"(b,c)
),pl"r"(a,pl"r"(c,b)),pl"r"(pl"r"(a,c),b),asspl1"r"(a,b
,c),ispl2"r"(pl"r"(b,c),pl"r"(c,b),a,compl"r"(b,c)),ass
pl2"r"(a,c,b)):is"r"(pl"r"(pl"r"(a,b),c),pl"r"(pl"r"(a,
c),b))
d@t2:=tr3is(real,pl"r"(pl"r"(a,b),pl"r"(c,d)),pl"r"(pl"
r"(pl"r"(a,b),c),d),pl"r"(pl"r"(pl"r"(a,c),b),d),pl"r"(
pl"r"(a,c),pl"r"(b,d)),asspl2"r"(pl"r"(a,b),c,d),ispl1"
r"(pl"r"(pl"r"(a,b),c),pl"r"(pl"r"(a,c),b),d,t1),asspl1
"r"(pl"r"(a,c),b,d)):is"r"(pl"r"(pl"r"(a,b),pl"r"(c,d))
,pl"r"(pl"r"(a,c),pl"r"(b,d)))
t3:=tris(real,mn"r"(pl"r"(a,b),pl"r"(c,d)),pl"r"(pl"r"(
```

AUTOMATH formalization of Gründlagen

I later learned that the “someone” was Lambertus Salomon (Bert) van Benthem Jutting, who used the AUTOMATH proof checker in the 1970s. The source code was preserved by Freek Wiedijk who also re-implemented AUTOMATH. We can still verify (instantly!) Landau’s book today.

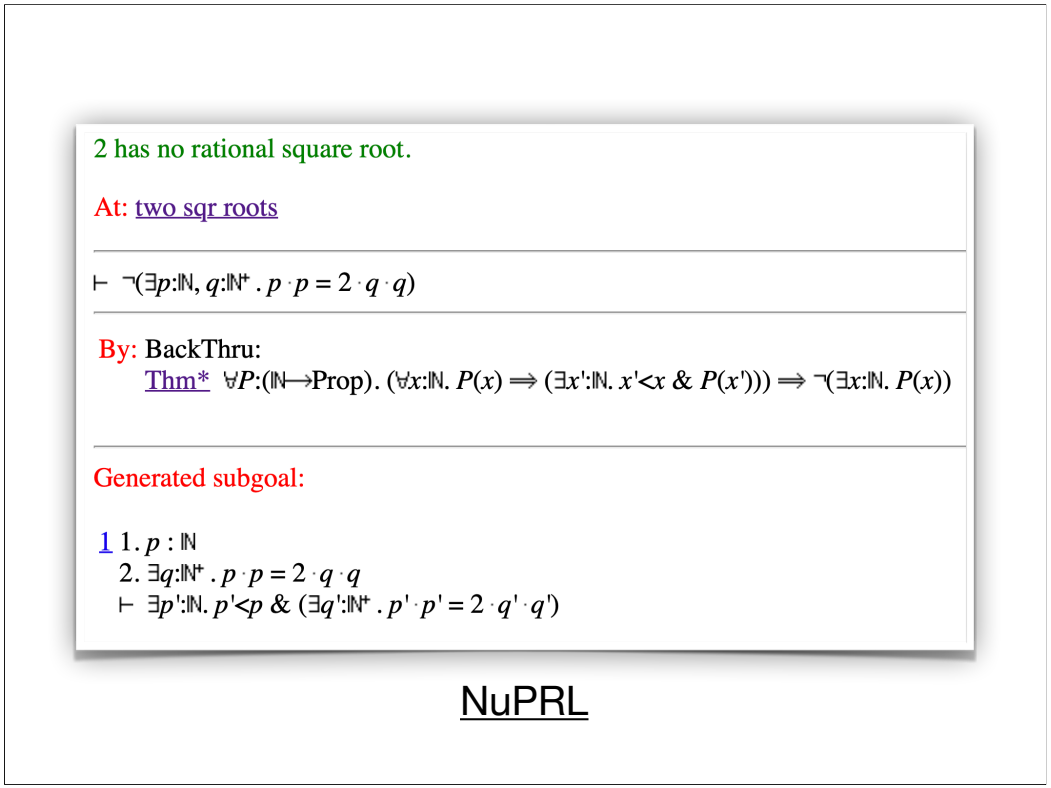
On the right you see the formal language of AUTOMATH. It was indeed a heroic and somewhat insane undertaking. The top line states that multiplication is associative. It is recognizable as a mathematical statement, albeit rather unfriendly. The AUTOMATH was revolutionary and ahead of its time, but could not realistically be used on a large scale. So what happened next?

```
:: A Borsuk Theorem on Homotopy Types
:: by Andrzej Trybulec
::
:: Received August 1, 1991
:: Copyright (c) 1991-2019 Association of Mizar Users


:: Topological preliminaries
theorem Th1: :: BORSUK_1:1
 for X being TopStruct
 for Y being SubSpace of X holds the carrier of Y c= the
proof end;

definition
 let X, Y be non empty TopSpace;
 let F be Function of X,Y;
 redefine attr F is continuous means :: BORSUK_1:def 1
 for W being Point of X
 for G being a_neighborhood of F . W ex H being a_neighbo
 compatibility
 proof end;
end;

:: deftheorem defines continuous BORSUK_1:def 1 :
```

Mizar

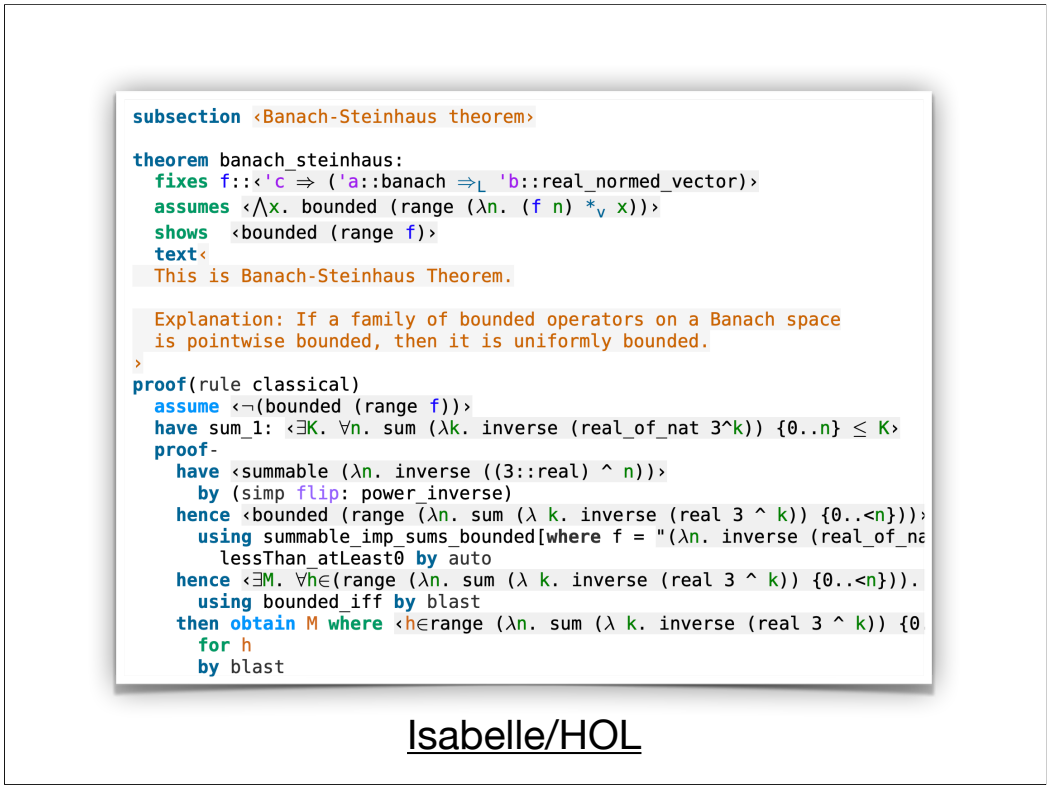
Computer scientists and computer-science minded logicians took up the task of making formalized mathematics practical. The idea of a *proof assistants* emerged: a program which assists a human in formalization. Over the decades various designs were explored.

Here we see fragments of code from some of the most well known proof assistants. Notice that they are a great deal friendlier than the original AUTOMATH.



Computer scientists and logicians took up the task of improving the formalism and the programs. The idea of a *proof assistants* emerged: a program which assists a human in formalization. Over decades various designs were explored.

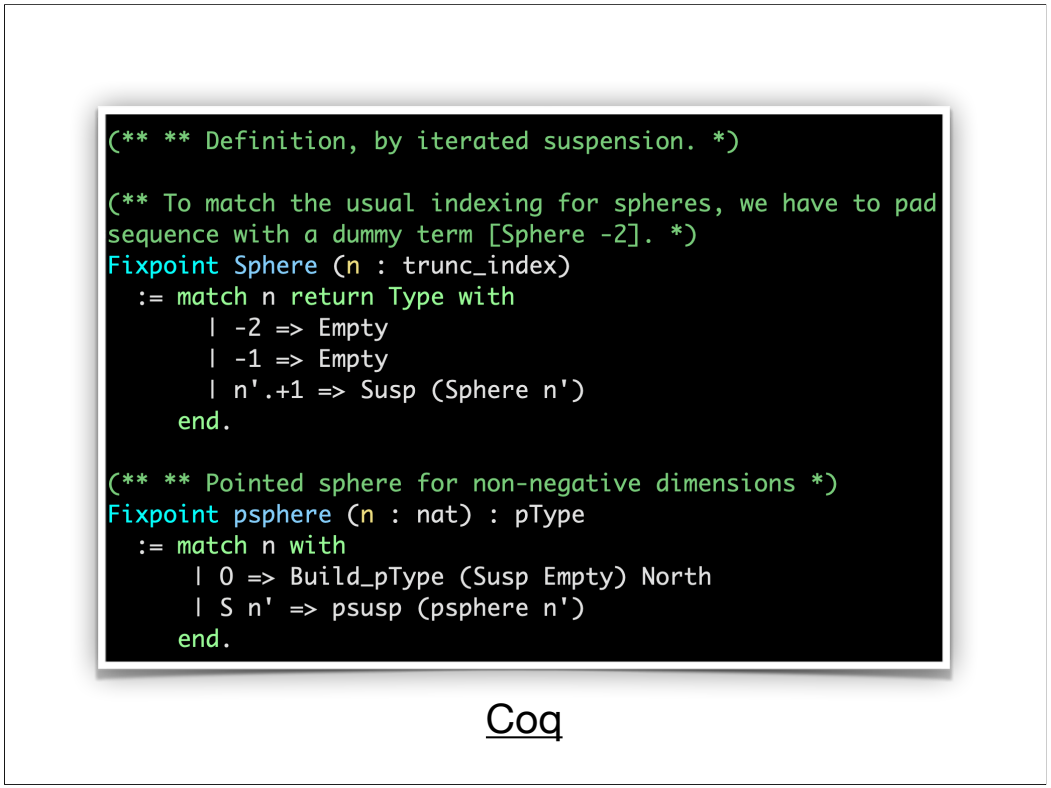
Here we see fragments of code from some of the most well known proof assistants. Notice that they are a great deal friendlier than the original AUTOMATH but still a bit intimidating, unless you're a programmer.



Isabelle/HOL

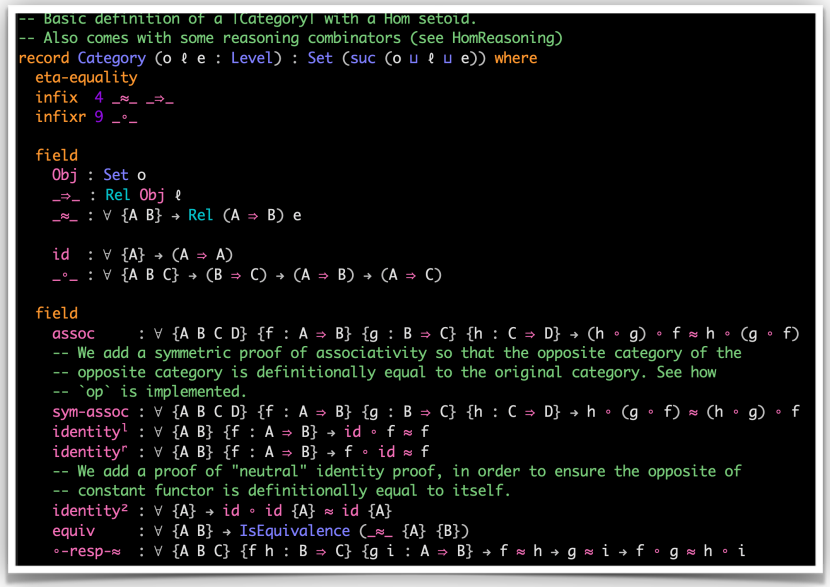
Computer scientists and logicians took up the task of improving the formalism and the programs. The idea of a *proof assistants* emerged: a program which assists a human in formalization. Over decades various designs were explored.

Here we see fragments of code from some of the most well known proof assistants. Notice that they are a great deal friendlier than the original AUTOMATH but still a bit intimidating, unless you’re a programmer.



Computer scientists and logicians took up the task of improving the formalism and the programs. The idea of a *proof assistants* emerged: a program which assists a human in formalization. Over decades various designs were explored.

Here we see fragments of code from some of the most well known proof assistants. Notice that they are a great deal friendlier than the original AUTOMATH but still a bit intimidating, unless you’re a programmer.



```
-- Basic definition of a 'Category' with a Hom setoid.
-- Also comes with some reasoning combinators (see HomReasoning)
record Category (o ℓ e : Level) : Set (suc (o ⊔ ℓ ⊔ e)) where
  eta-equality
  infix  4 _≈_ _⇒_
  infixr 9 _∘_

  field
    Obj : Set o
    _⇒_ : Rel Obj ℓ
    _≈_ : ∀ {A B} → Rel (A ⇒ B) e

    id  : ∀ {A} → (A ⇒ A)
    _∘_ : ∀ {A B C} → (B ⇒ C) → (A ⇒ B) → (A ⇒ C)

  field
    assoc     : ∀ {A B C D} {f : A ⇒ B} {g : B ⇒ C} {h : C ⇒ D} → (h ∘ g) ∘ f ≈ h ∘ (g ∘ f)
    -- We add a symmetric proof of associativity so that the opposite category of the
    -- opposite category is definitionally equal to the original category. See how
    -- `op` is implemented.
    sym-assoc : ∀ {A B C D} {f : A ⇒ B} {g : B ⇒ C} {h : C ⇒ D} → h ∘ (g ∘ f) ≈ (h ∘ g) ∘ f
    identityˡ : ∀ {A B} {f : A ⇒ B} → id ∘ f ≈ f
    identityʳ : ∀ {A B} {f : A ⇒ B} → f ∘ id ≈ f
    -- We add a proof of "neutral" identity proof, in order to ensure the opposite of
    -- constant functor is definitionally equal to itself.
    identity² : ∀ {A} → id ∘ id {A} ≈ id {A}
    equiv     : ∀ {A B} → IsEquivalence (_≈_ {A} {B})
    ∘-resp-≈  : ∀ {A B C} {f h : B ⇒ C} {g i : A ⇒ B} → f ≈ h → g ≈ i → f ∘ g ≈ h ∘ i
```

Agda

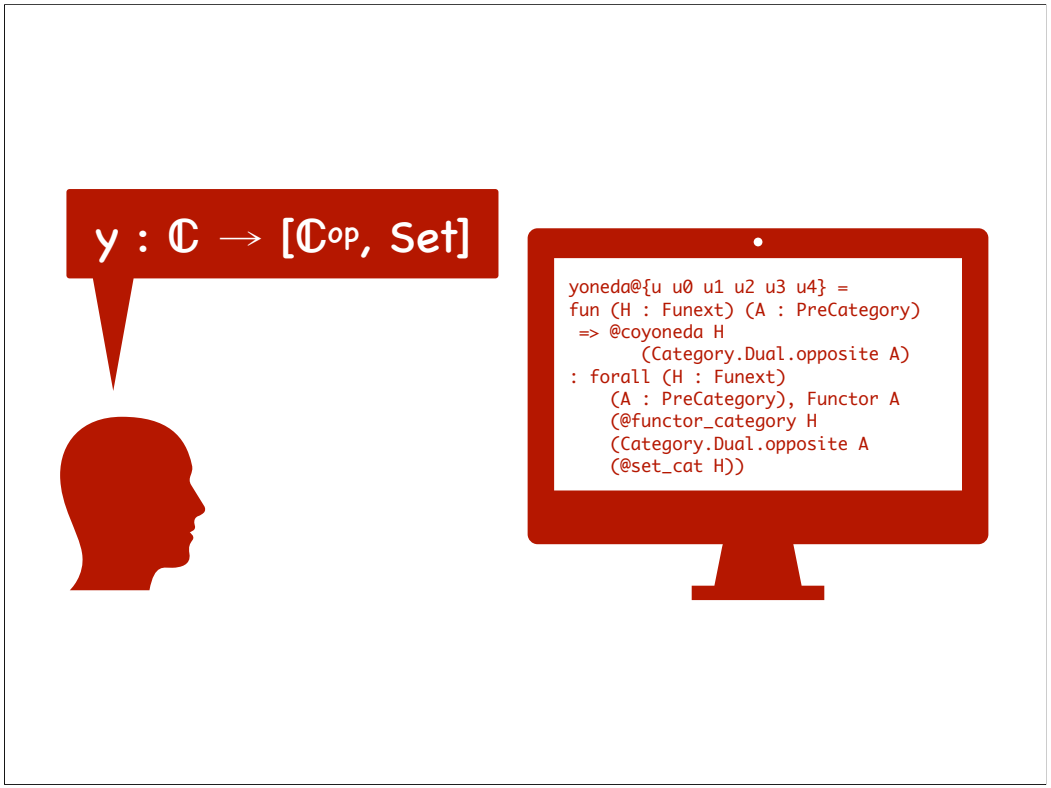
Computer scientists and logicians took up the task of improving the formalism and the programs. Over decades various designs were explored, and the idea of a *proof assistants* emerged: a program which assists a human in formalization.

Here we see fragments of code from some of the most well known proof assistants. Notice that they are a great deal friendlier than the original AUTOMATH but still a bit intimidating, unless you’re a programmer.

How does a proof assistant work?

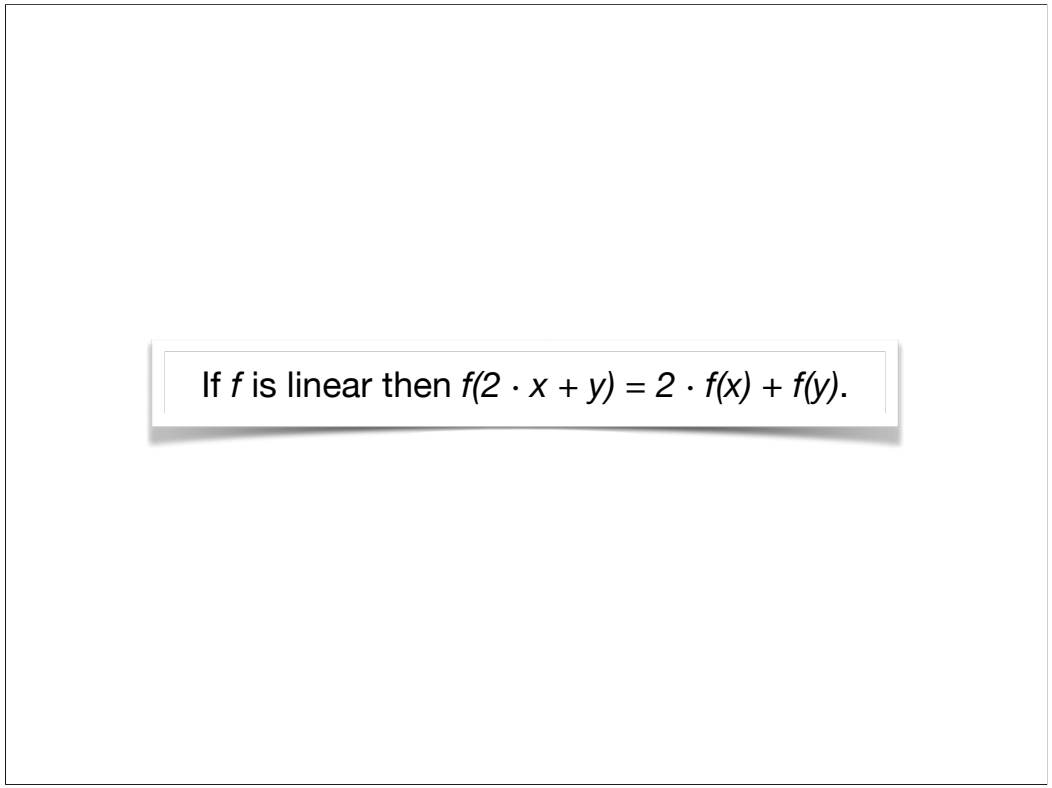
Allow me to explain a little bit what proof assistants are and how they work.

Since they are complex pieces of software there are many aspects of their design. We shall focus on those that are relevant and interesting to mathematicians.



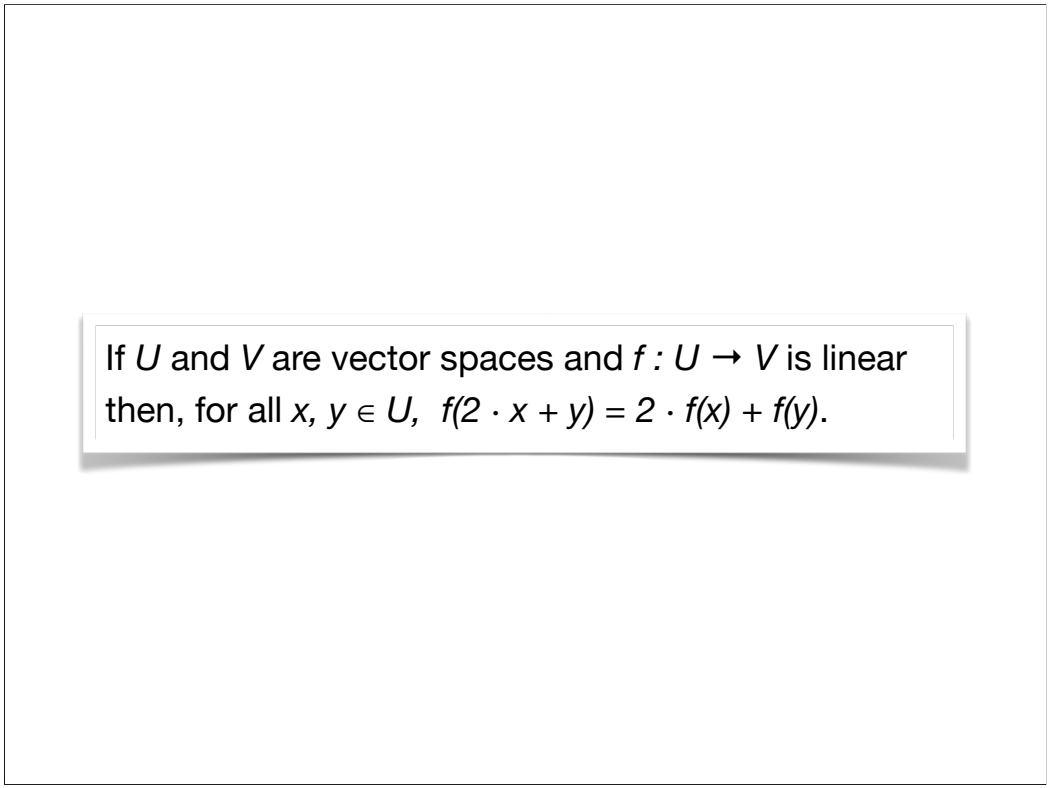
It has been clear since the days of AUTOMATH that the real challenge is not bare computer verification of formal proofs, but one of **human-computer interaction**: given the considerable gap between mathematics as done by humans and the formal mathematics understood by computers, how can the human and the machine cooperate?

To get some idea of what needs to be accomplished, let us look at an example.



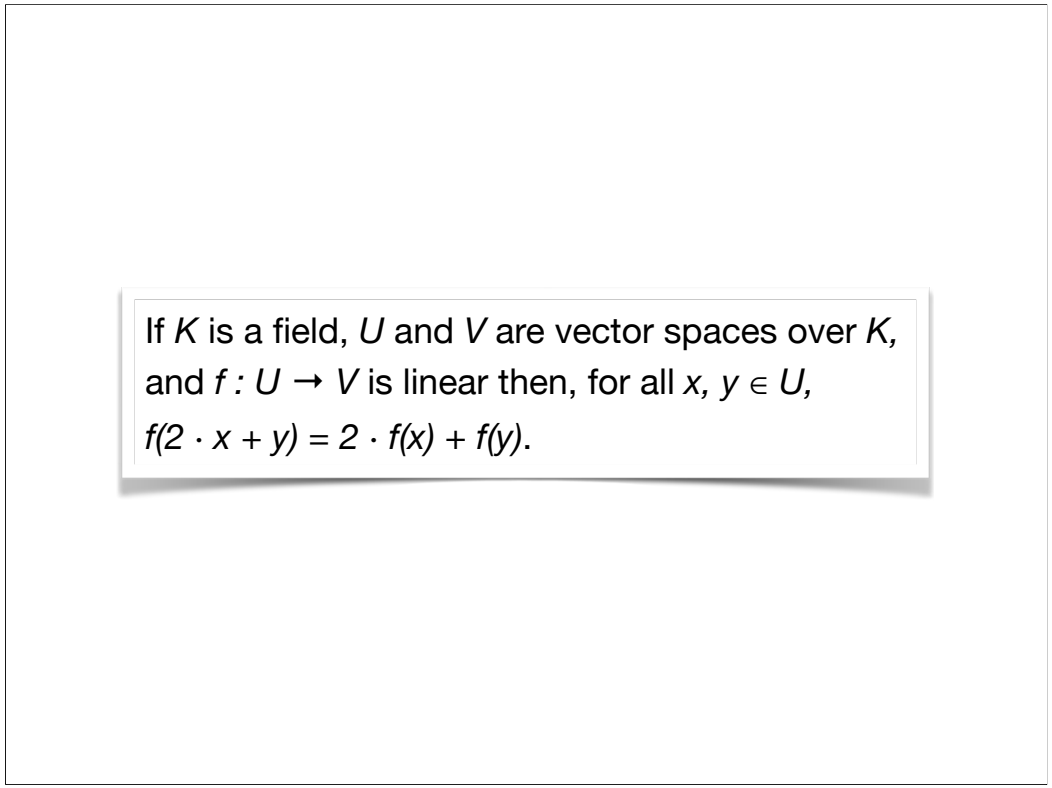
Read the statement. Do you understand it? Of course you do, you're a mathematician. You can also tell that the statement is obviously true.

Yet from a formal point of view, the statement is quite inexact, as it omits a number of details. The details do not matter and your brain can recover them automatically if so desired, but it is perhaps surprising how many details need to be filled in.

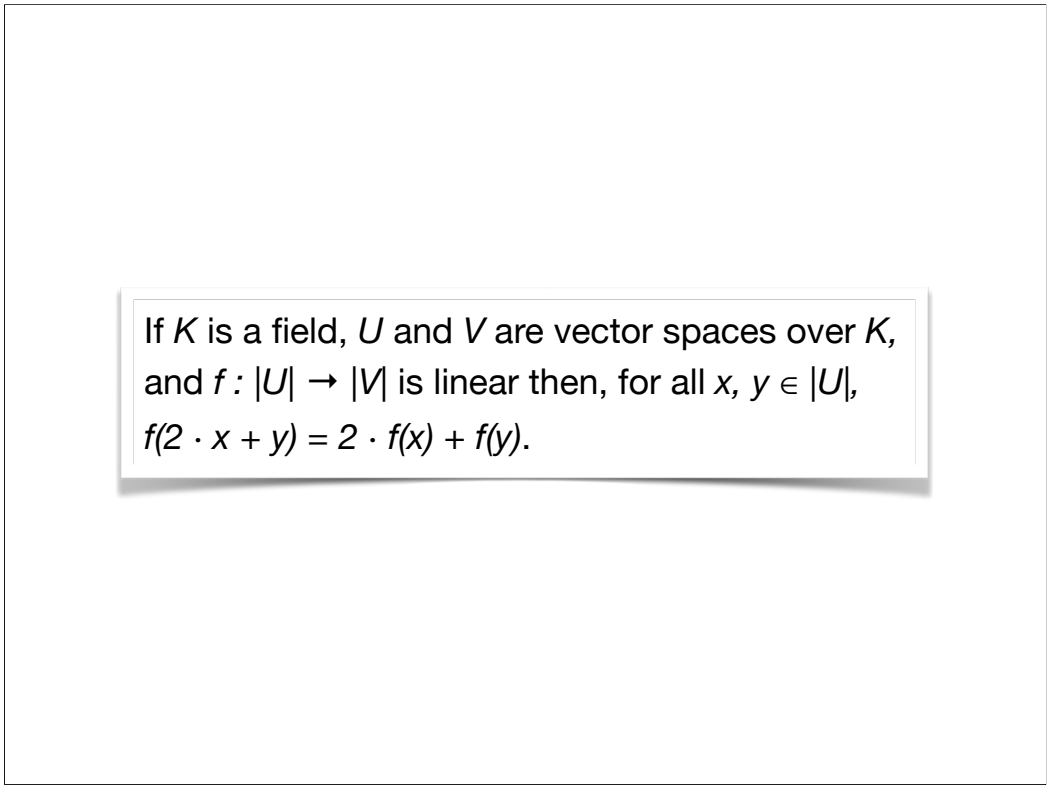


Firstly, we should make the domain and codomain of f explicit by introducing vector spaces U and V. Next, we can tell that x and y range over U, or else f(x) and f(y) makes no sense.

It is understood that U and V are vector spaces over some unmentioned field.



Ok, so let us make the field explicit as well.
See what is going on? As the statement gets more precise, it gets harder to understand and its gist is lost in the bureaucracy. There is a reason people prefer concise imprecise statements to precise obfuscated ones. And we are still not done.

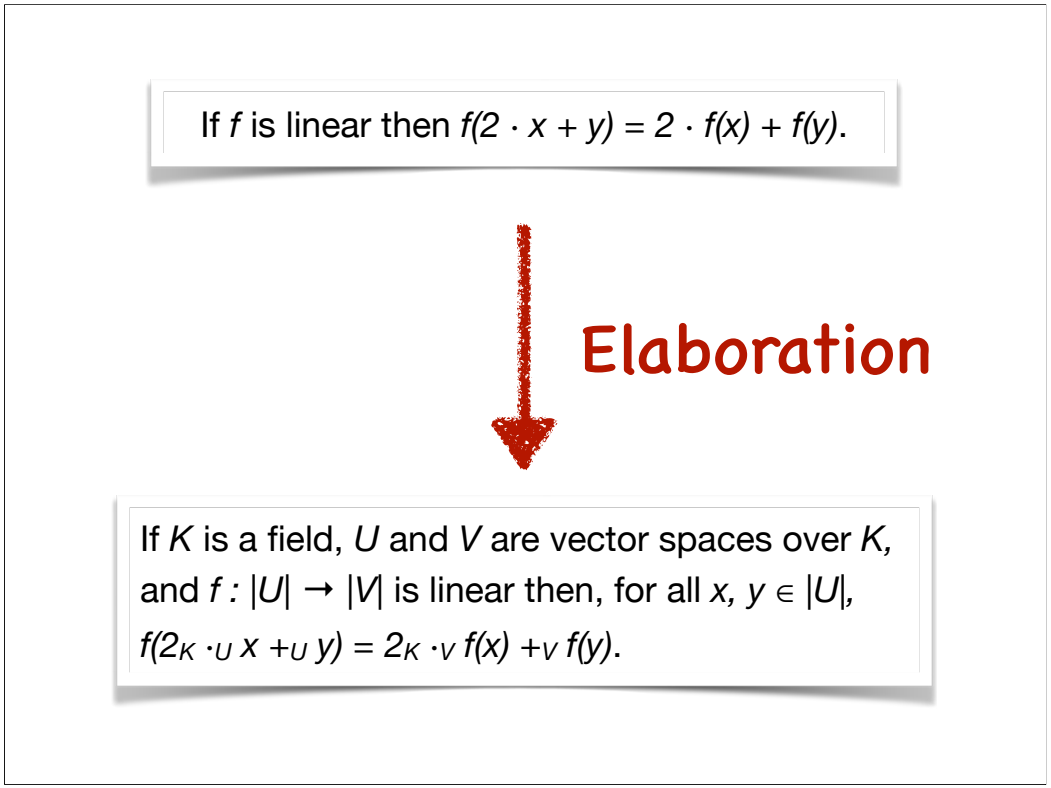


A vector space U is a *structure* with an underlying *carrier set* |U|. These should be distinguished.
(Side remark: regarding f, we could alternatively state that it is a morphism in the category of vector spaces, but then f would not be a map *together* with the fact that it is linear, so we would have to write something like |f| for the underlying map and write |f|(x) instead of f(x).)

The numeral 2, is that a natural number? It should be an element of K. And why are we using + for addition in both U and V?

If K is a field, U and V are vector spaces over K, and $f : |U| \to |V|$ is linear then, for all $x, y \in |U|$, $f(2_K \cdot_U x +_U y) = 2_K \cdot_V f(x) +_V f(y)$.

Here is the final form, although we could probably go on.
We tagged 2 with K to indicate that we mean the image of 2 by the canonical map $\mathbb{Z} \to K$.
We also tagged vector addition and scalar multiplication with U and V, respectively.



The process we just described is called **elaboration**.

It is part of *implicit* mathematical knowledge that is passed on by observation and imitation, and rarely talked about. The 20th century logic has mostly ignored elaboration, because it preoccupied itself with other problems, that were more relevant at the time.

Elaboration

Informal	Formal
“An assumed ambient field K.”	existential variable
“Guess that x ranges over U.”	type inference
“Automatically change U to |U|.”	implicit coercion
“Read 2 as element of K.”	notational scope

Elaboration is an essential part of modern proof assistants. It is the bridge between the human and the machine. It was not developed by mathematicians or logicians, but by computer scientists who faced similar issues of human-computer interaction when designing programming languages.

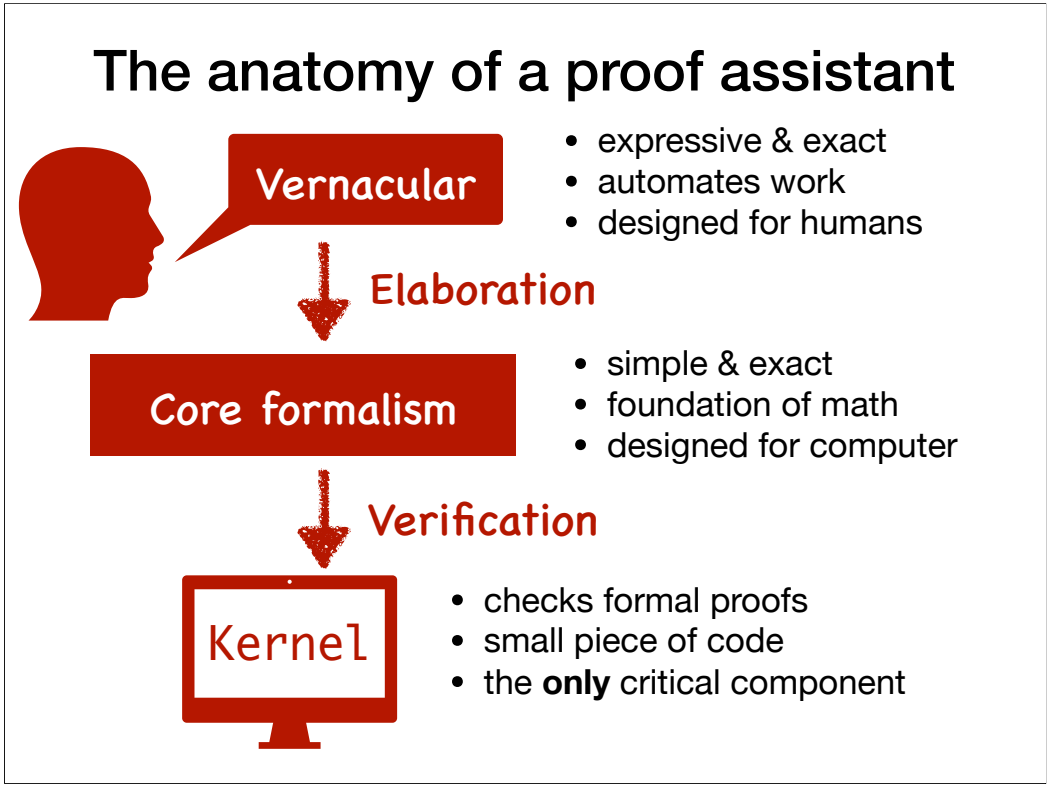
When you learn to use your first proof assistant, keep in mind that you already know the elaboration techniques, you just never gave them names or thought of them mathematically.

```
import linear_algebra

variables
  (K : Type*) [field K]
  (U : Type*) [add_comm_group U] [module K U]
  (V : Type*) [add_comm_group V] [module K V]
  (f : U →ₗ[K] V)

example: ∀ x y, f (2 • x + y) = 2 • f x + f y
  := by simp
```

Lean

You might be curious how the statement would be written in a proof assistant. Here it is in the syntax of the Lean proof assistant. There are some mysterious bits, but for the most part it is comprehensible, and it certainly is not worse than LaTeX. The last line, “by simp”, is how you tell Lean to prove the statement using the tactic “simp” (which stand for “simple”).

I shall say more about Lean and its rapidly growing community at the end of the talk.



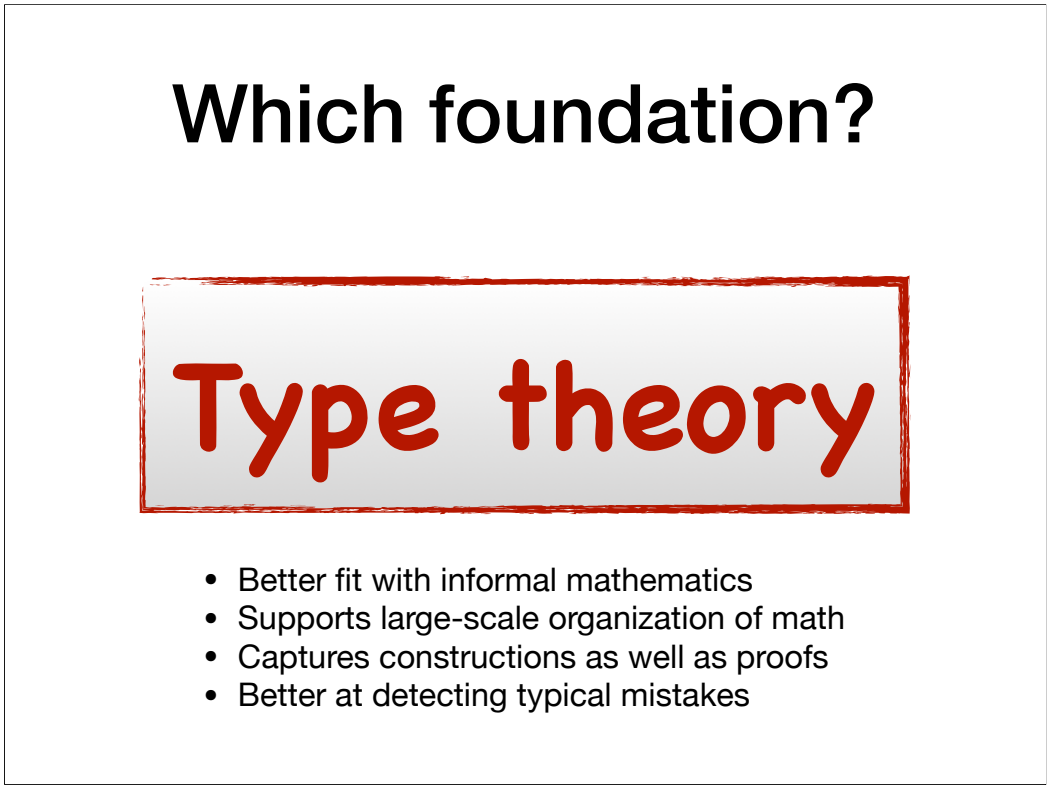
Having explained elaboration, let us look at how a modern proof assistant is structured.
There are *two* formal languages: the vernacular and the core formalism.

The vernacular is designed for humans, while still formally precise so that computers can process it. It supports common mathematical notations, provides support for automation, and generally attempts to accommodate the user. This is what you need to learn to master a proof assistant.

The elaborator translates the vernacular to a core formalism. This is the underlying formal mathematical foundation as commonly understood by mathematicians and logicians. It should be mathematically well-understood and free of unnecessary complexity.

The verification is carried out by a special component of the proof assistant, the kernel. This is the **only** critical component: a bug in the elaborator is just annoying, whereas a bug in the kernel may lead to faulty mathematics. The kernel is designed to be small so that it can be more easily audited and trusted.

Further reading: Robert Pollack, “How to believe a Machine-Checked Proof” (1996)



We have not yet discussed which mathematical foundation should be used.

It is an accepted norm that set theory is the foundation of mathematics. Indeed, in the 20th century set theory helped relate and connect all branches of mathematics, and served as a lingua franca.

However, most proof assistants do not use set theory. Instead they rely on various incarnations of type theory. We may speculate why this is the case, but it is undeniable that in practice type theory is the preferred formalism.

Further reading: Andrej Bauer, "What makes dependent type theory more suitable than set theory for proof assistants?", an answer to MathOverflow question (2020)

Why formalize?

I have not yet addresses the obvious question:
Why bother with formalization of mathematics at all?

People have been drawn to formalized mathematics for various reasons.

Checking inhumane proofs

1. Four-color theorem [Gonthier et al., 2005]

   Verified in Coq, 633 critical configurations.

2. Kepler's sphere packing conjecture [Hales et al., 2014]

   Verified in HOL-light, 23 000 non-linear inequalities.

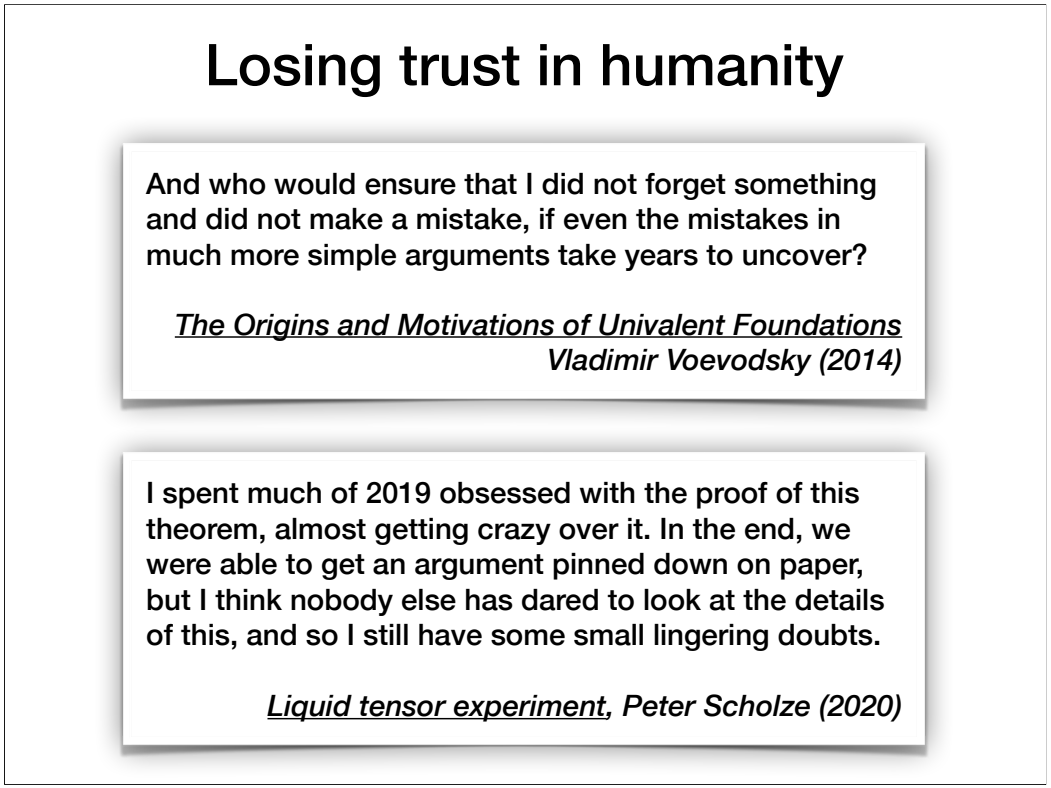
3. CompCert: verified C compiler [Leroy et al., 2009]

   10000+ theorems about C compiler & PowerPC, ARM, RISC-V and x86 processors.

Sometimes formalization is necessary because the proof is not realistically checkable by hand.

Two famous examples from mathematics are the four-color theorem and Kepler's conjecture. At first both were checked with computer programs that performed large amounts of computation – but nobody proved that those programs were free of mistakes (which they were not). This has since been rectified: all the theorems, proofs, as well as programs were formalized in proof assistants and are water-tights. We are talking about a level of confidence that is far above that of a proof published in a reputable math journal.

An important motivation for formalized mathematics comes from computer science. Security and reliability of computer systems can be established by formal verification of proofs. In a typical situation we have to prove thousands of boring theorems, which is a hopeless task without machine support. A noteworthy example is CompCert – an optimizing C compiler that has been formally proven to work correctly.



Some mathematicians have reached a level of complexity in their work at which they have lost confidence in their own ability to carry out the proofs correctly and to spot mistakes. Where else would they turn for help, if not to machines?

Or to put it another way: to break the one-mind barrier, mathematicians will organize themselves into hives, centered around machines.

Further reading: Jacques Carette et al., "Big Math and the One-Brain Barrier A Position Paper and Architecture Proposal" (2019)

Creating new mathematics

Although such a formalization is not part of this book, much of the material presented here was actually done first in the fully formalized setting inside a proof assistant, and only later "unformalized" to arrive at the presentation you find before you — a remarkable inversion of the usual state of affairs in formalized mathematics.

Homotopy type theory: Univalent foundations of mathematics
Univalent foundations programme (2013)

Formalized mathematics is not only about checking what we already know. It also leads to new discoveries. Those who have formalized mathematics will tell you that it often leads to significant improvements and simplifications, and sometimes to new discoveries.

In 2013, at the Institute for Advanced Study in Princeton, a group of mathematicians and computer scientists was developing a new foundation of mathematics, known as *homotopy type theory*. It was so new & unfamiliar that we did not know how to communicate and think in it. We turned to proof assistants for guidance, and "unformalized" homotopy type theory once we learned how it works. The result was "HoTT book", 600+ pages written in 6 months by two dozen mathematicians.

Recognizing the potential

I now clearly understand that software such as Lean is part of the inevitable future of mathematics. … Tools such as Lean will one day help us mathematicians search for theorems in the literature, and help us to prove theorems. … These tools may also change the way we teach. … It is possible that they will begin to do research semi-autonomously, perhaps uncover problems in the literature. Maybe they will replace research mathematicians.

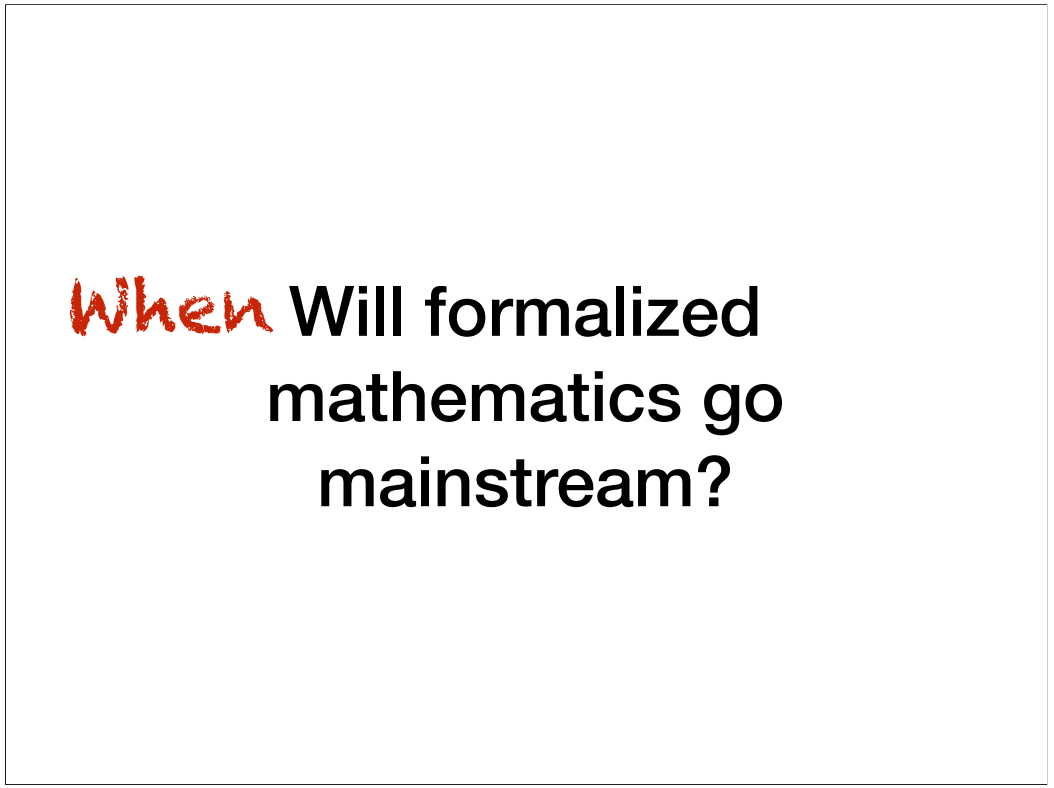
The future of mathematics?
Kevin Buzzard (2019)

If a machine can can beat the best chess and go players, does it not stand to reason that in the future it will also help professional mathematicians?

Some mathematicians are recognizing the potential & significance of doing mathematics with computers, and are not willing to sit and wait for computer scientists to do work for them.

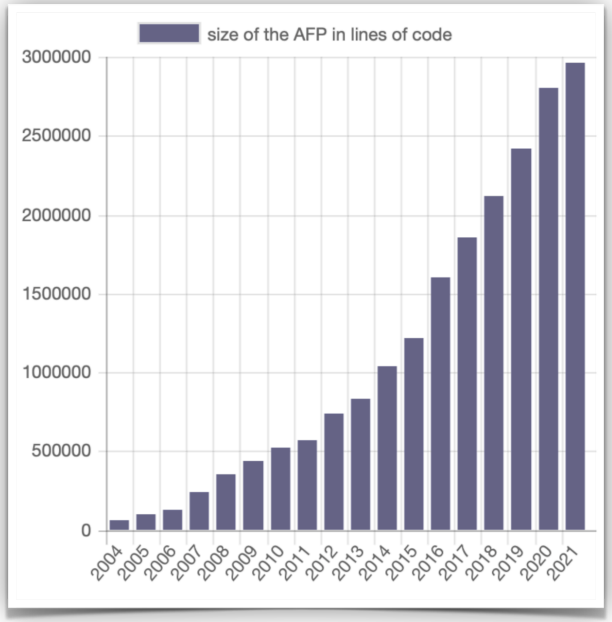
Will formalized mathematics go mainstream?

To wrap up, let me speculate about the answer to the obvious question. It is **not** “whether” formalized mathematics transform mathematics …



… but rather **when** the transformation will take place.

It would be preposterous to expect that mathematics is immune from the disruptive power of computer technology.



Archive of Formal Proofs

The growth of formalized mathematics in the last two decades has been impressive. We have progressed from small groups of enthusiastic researchers to a rapidly growing community which is increasingly shifting from computer scientists to mathematicians.

Of course, formalized mathematics is still just a tiny part of all of mathematics. However, it is not its size, but the potential for disruption that is relevant.

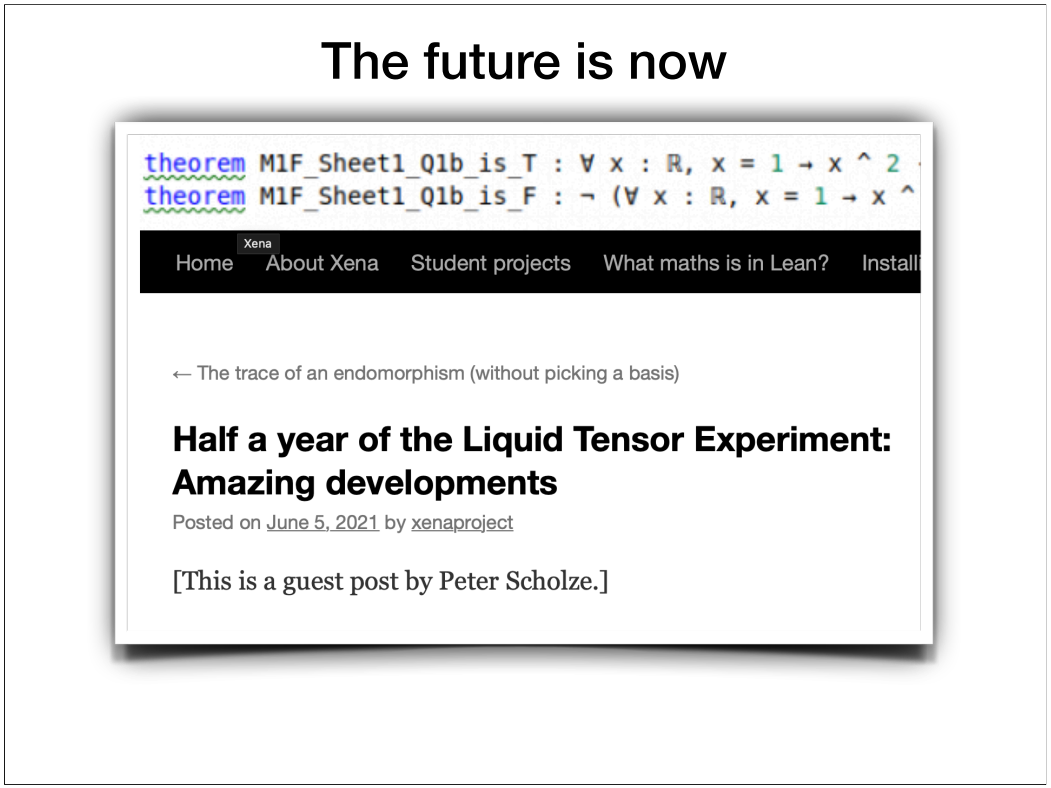
Lean mathematical library

To give you an idea of the collaborative power of formalized mathematics, let us watch a short video. It shows mathematicians working on the mathematical library “mathlib”, implemented in the Lean proof assistant.

The moving figures are people, the tree shows the directory structure and the files of the library. The rays of light indicate a user modifying a file.

Ok, it’s pretty, but does it tell us anything? Yes. We are not watching separate groups of two or three mathematicians, each belaboring on a paper of their own. This is not a medieval mathematicians’ guild (which is how mathematics is still organized today), but an industrial division of labor. It is a math hive. It is exciting and new.

Separate link to video: https://vimeo.com/566990363



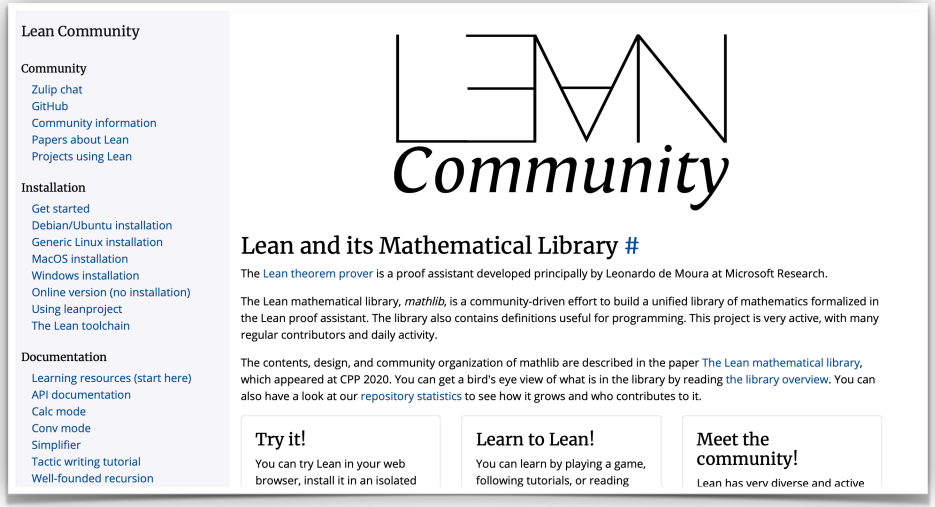
As I was preparing the slides for this talk, an article in Nature took notice of a recent success in formalized mathematics. It is worth reporting here.

Half a year ago Peter Scholze posed a formalization challenge: formalize a difficult new theorem that he was not sure about. The challenge took half a year, which is comparable with refereeing process, but the results are far more significant: we **know** that the theorem is true beyond reasonable doubt, and new mathematics was discovered during formalization. The is a glimpse of the future – true cooperation of man & machine.

Further reading:

- Peter Scholze: Liquid tensor experiment
- Peter Scholze: Half a year of the Liquid Tensor Experiment: Amazing developments
- Nature: Mathematicians welcome computer-assisted proof in ‘grand unification’ theory

How do I join?



If you are interested in trying out & learning about formalized mathematics, I recommend the new kind on the block – the Lean proof assistant and its mathematical library. The community, spearheaded by Kevin Buzzard, has thousands of members. They are a friendly bunch who takes care of beginners.

Of course, you can also try out the older, venerable proof assistants, especially if you are interested in computer-science applications, constructive mathematics, or homotopy type theory.

Thank you!

This material is based upon work supported by the Air Force Office of Scientific Research under award number FA9550-21-1-0024.