Something to Lean on; fun with interactive theorem provers

Alex J. Best

BU Math Retreat 2019

The problem

Mathematicians make mistakes.

The problem

Mathematicians make mistakes.

Sometimes they publish these mistakes.

The problem

Mathematicians make mistakes.

Sometimes they publish these mistakes.

Sometimes nobody notices.

The problem

Mathematicians make mistakes.

Sometimes they publish these mistakes.

Sometimes nobody notices.

At least for a while...

The problem

Mathematicians make mistakes.

Sometimes they publish these mistakes.

Sometimes nobody notices.

At least for a while...

However this uncertainty takes up time and energy, what if referees only needed to judge the importance, novelity and quality of exposition, not check the arguments.

What is proof

What is proof

On an abstract level proof is a rigid mathematical concept.

What is proof

On an abstract level proof is a rigid mathematical concept.

In reality it is a social construct.

What is proof

On an abstract level proof is a rigid mathematical concept.

In reality it is a social construct.

> *This story got me scared. Starting from 1993 multiple groups of mathematicians studied the "Cohomological Theory" paper at seminars and used it in their work and none of them noticed the mistake.*
>
> *And it clearly was not an accident. A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct, is hardly ever checked in detail.*
>
> *— Vladimir Voevodsky*

Some examples: Grunwald(-Wang) and K-theory

Some days later I was with Artin in his office when Wang appeared. He said he had a counterexample to a lemma which had been used in the proof. An hour or two later, he produced a counterexample to the theorem itself... Of course he [Artin] was astonished, as were all of us students, that a famous theorem with two published proofs, one of which we had all heard in the seminar without our noticing anything, could be wrong.

— Tate

Some examples: Grunwald(-Wang) and K-theory

Some days later I was with Artin in his office when Wang appeared. He said he had a counterexample to a lemma which had been used in the proof. An hour or two later, he produced a counterexample to the theorem itself... Of course he [Artin] was astonished, as were all of us students, that a famous theorem with two published proofs, one of which we had all heard in the seminar without our noticing anything, could be wrong.

— Tate

The groundbreaking 1986 paper "Algebraic Cycles and Higher K-theory" by Spencer Bloch was soon after publication found by Andrei Suslin to contain a mistake in the proof of Lemma 1.1. The proof could not be fixed.

— Voevodsky

The solution

Work at an extremely high level of rigour, with every statement logical step spelled out in extreme detail and checked by multiple independent people.

The solution

Work at an extremely high level of rigour, with every statement logical step spelled out in extreme detail and checked by multiple independent people.

Problem
This is really boring!

The solution

Work at an extremely high level of rigour, with every statement logical step spelled out in extreme detail and checked by multiple independent people.

Problem
This is really boring!

> *But to do the work at the level of rigor and precision I felt was necessary would take an enormous amount of effort and would produce a text that would be very difficult to read. And who would ensure that I did not forget something and did not make a mistake, if even the mistakes in much more simple arguments take years to uncover?*
>
> — *Voevodsky*

The solution: Take 2

What if computers could do the boring work for us?

The solution: Take 2

What if computers could do the boring work for us?

Computers are:

- Capable of checking basic logical statements,

The solution: Take 2

What if computers could do the boring work for us?

Computers are:

- Capable of checking basic logical statements,
- Fast,

The solution: Take 2

What if computers could do the boring work for us?

Computers are:

- Capable of checking basic logical statements,
- Fast,
- Never complain.

The new problem

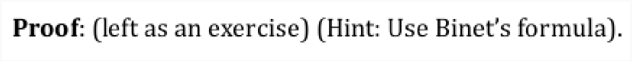
How do you describe the steps of a proof to a computer with as little pain as possible?

The new problem

How do you describe the steps of a proof to a computer with as little pain as possible? Often mathematicians leave unsaid many steps which are intuitive or easily supplied.

The new problem

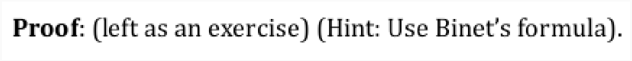
How do you describe the steps of a proof to a computer with as little pain as possible? Often mathematicians leave unsaid many steps which are intuitive or easily supplied.

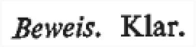


Proof: (left as an exercise) (Hint: Use Binet's formula).

The new problem

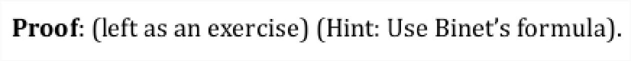
How do you describe the steps of a proof to a computer with as little pain as possible? Often mathematicians leave unsaid many steps which are intuitive or easily supplied.

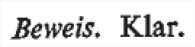

Proof: (left as an exercise) (Hint: Use Binet's formula).

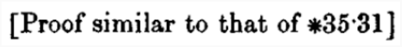

Beweis. **Klar.**

The new problem

How do you describe the steps of a proof to a computer with as little pain as possible? Often mathematicians leave unsaid many steps which are intuitive or easily supplied.

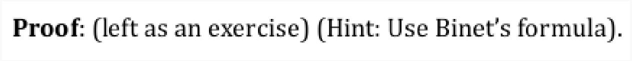

Proof: (left as an exercise) (Hint: Use Binet's formula).

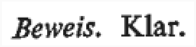

Beweis. Klar.

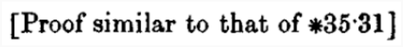

[Proof similar to that of ∗35·31]

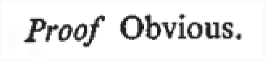
The new problem

How do you describe the steps of a proof to a computer with as little pain as possible? Often mathematicians leave unsaid many steps which are intuitive or easily supplied.


Proof: (left as an exercise) (Hint: Use Binet's formula).

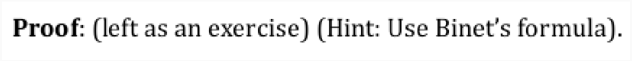

Beweis. Klar.

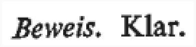

[Proof similar to that of ∗35·31]

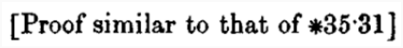

Proof Obvious.

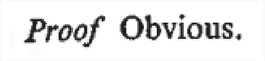
The new problem

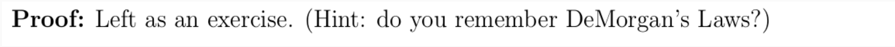
How do you describe the steps of a proof to a computer with as little pain as possible? Often mathematicians leave unsaid many steps which are intuitive or easily supplied.


Proof: (left as an exercise) (Hint: Use Binet's formula).


Beweis. Klar.

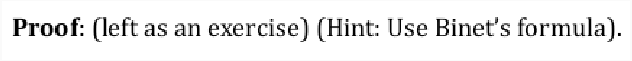

[Proof similar to that of ∗35·31]

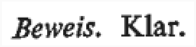

Proof Obvious.

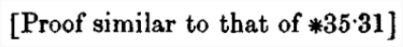

Proof: Left as an exercise. (Hint: do you remember DeMorgan's Laws?)

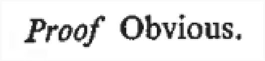
The new problem

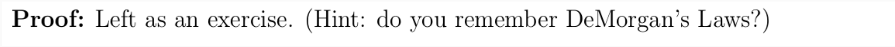
How do you describe the steps of a proof to a computer with as little pain as possible? Often mathematicians leave unsaid many steps which are intuitive or easily supplied.

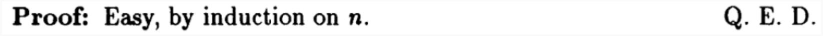

Proof: (left as an exercise) (Hint: Use Binet's formula).


Beweis. **Klar.**


[Proof similar to that of ∗35·31]

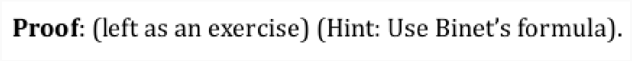

Proof Obvious.

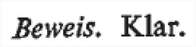

Proof: Left as an exercise. (Hint: do you remember DeMorgan's Laws?)

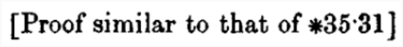

Proof: Easy, by induction on *n*. Q. E. D.

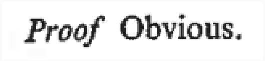
The new problem

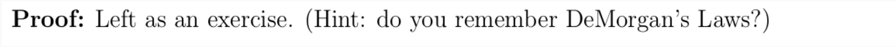
How do you describe the steps of a proof to a computer with as little pain as possible? Often mathematicians leave unsaid many steps which are intuitive or easily supplied.

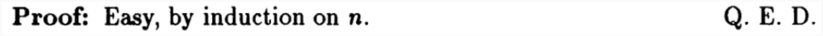

Proof: (left as an exercise) (Hint: Use Binet's formula).


Beweis. Klar.


[Proof similar to that of ∗35·31]


Proof Obvious.


Proof: Left as an exercise. (Hint: do you remember DeMorgan's Laws?)


Proof: Easy, by induction on n. Q. E. D.

The computer will probably not understand these, but in order to stay sane we must strike a balance between detail and verbosity.

Enter Lean

The idea of trying this sort of thing has been around for a while.

Enter Lean

The idea of trying this sort of thing has been around for a while.

But only within the last few years has it begun to seem more feasible for an average mathematician to do this. Tools have gotten better, slowly this idea has gained traction.

Enter Lean

The idea of trying this sort of thing has been around for a while.

But only within the last few years has it begun to seem more feasible for an average mathematician to do this. Tools have gotten better, slowly this idea has gained traction.

Recently an interactive proof assistant called Lean has been under heavy development. And I've been playing with it.

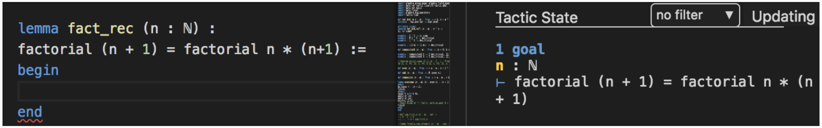
Let me show you some lean code:

```
lemma fact_rec (n : ℕ) :
factorial (n + 1) = factorial n * (n+1) :=
begin
-- write out the definition of factorial
unfold factorial,
-- remember {1,...,n+1} = {1,...,n} ∪ {n+1}
rewrite list.range'_concat 1 n,
-- the product of two sequences joined together is
    just the product of the products of each sequence
rewrite list.prod_append,
-- I'm bored already are we done here?
simp,
-- YES!
end
```

```
lemma fact_rec (n : ℕ) :
factorial (n + 1) = factorial n * (n+1) :=
begin
-- write out the definition of factorial
unfold factorial,
-- remember {1,...,n+1} = {1,...,n} ∪ {n+1}
rewrite list.range'_concat 1 n,
-- the product of two sequences joined together is
   just the product of the products of each sequence
rewrite list.prod_append,
-- I'm bored already are we done here?
simp,
-- YES!
end
```

We can replace all of the above with: `by unfold factorial; simp [list.range'_concat, list.prod_append]`

(not so?) Live demo!

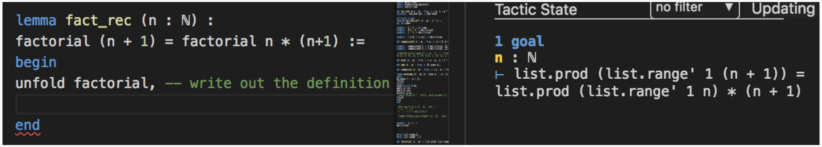


```
lemma fact_rec (n : ℕ) :
factorial (n + 1) = factorial n * (n+1) :=
begin

end
```

```
Tactic State
1 goal
n : ℕ
⊢ factorial (n + 1) = factorial n * (n
+ 1)
```

(not so?) Live demo!

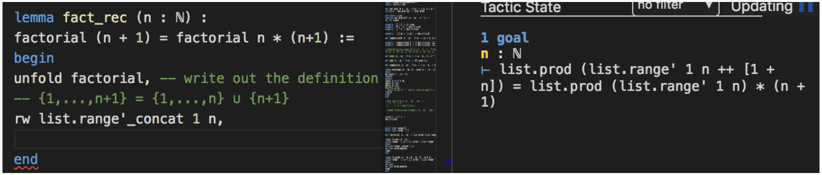


```
lemma fact_rec (n : ℕ) :
factorial (n + 1) = factorial n * (n+1) :=
begin
unfold factorial, -- write out the definition

end
```

```
1 goal
n : ℕ
⊢ list.prod (list.range' 1 (n + 1)) =
list.prod (list.range' 1 n) * (n + 1)
```

(not so?) Live demo!

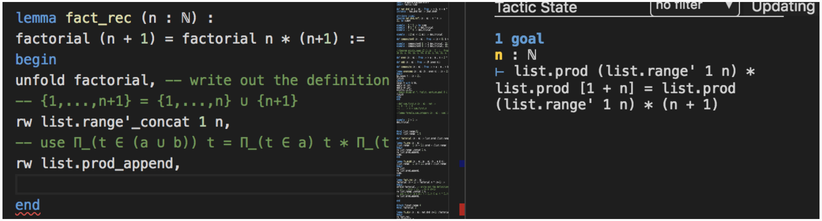


```
lemma fact_rec (n : ℕ) :
factorial (n + 1) = factorial n * (n+1) :=
begin
unfold factorial, -- write out the definition
-- {1,...,n+1} = {1,...,n} ∪ {n+1}
rw list.range'_concat 1 n,

end
```

```
1 goal
n : ℕ
⊢ list.prod (list.range' 1 n ++ [1 +
n]) = list.prod (list.range' 1 n) * (n +
1)
```

(not so?) Live demo!

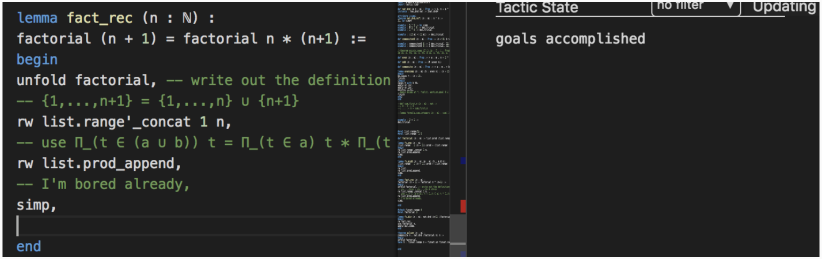


```
lemma fact_rec (n : ℕ) :
factorial (n + 1) = factorial n * (n+1) :=
begin
unfold factorial, -- write out the definition
-- {1,...,n+1} = {1,...,n} ∪ {n+1}
rw list.range'_concat 1 n,
-- use Π_(t ∈ (a ∪ b)) t = Π_(t ∈ a) t * Π_(t
rw list.prod_append,

end
```

```
1 goal
n : ℕ
⊢ list.prod (list.range' 1 n) *
list.prod [1 + n] = list.prod
(list.range' 1 n) * (n + 1)
```

(not so?) Live demo!



```
lemma fact_rec (n : ℕ) :
factorial (n + 1) = factorial n * (n+1) :=
begin
unfold factorial, -- write out the definition
-- {1,...,n+1} = {1,...,n} ∪ {n+1}
rw list.range'_concat 1 n,
-- use Π_(t ∈ (a ∪ b)) t = Π_(t ∈ a) t * Π_(t
rw list.prod_append,
-- I'm bored already,
simp,

end
```

```
goals accomplished
```

But can it do research?

Research level mathematics requires a vast body of knowledge to even think about.

But can it do research?

Research level mathematics requires a vast body of knowledge to even think about.

As humans we forget this and also gloss over things we (think we) know well.

But can it do research?

Research level mathematics requires a vast body of knowledge to even think about.

As humans we forget this and also gloss over things we (think we) know well.

For instance one can forget what a Dedekind cut is, or Peano arithmetic and still think about real and natural numbers without an issue. Our intuition allows us to abstract these concepts so far away that we don't have to work from the ground up when approaching a problem.

Perfectoids

Peter Scholze won a Fields medal in 2018 for "transforming arithmetic algebraic geometry over p-adic fields through his introduction of perfectoid spaces, with application to Galois representations, and for the development of new cohomology theories."

Perfectoids

Peter Scholze won a Fields medal in 2018 for “transforming arithmetic algebraic geometry over p-adic fields through his introduction of perfectoid spaces, with application to Galois representations, and for the development of new cohomology theories.” Within the past 10 years in number theory, this one definition above all others has revolutionised the way (some) people think.

Perfectoids

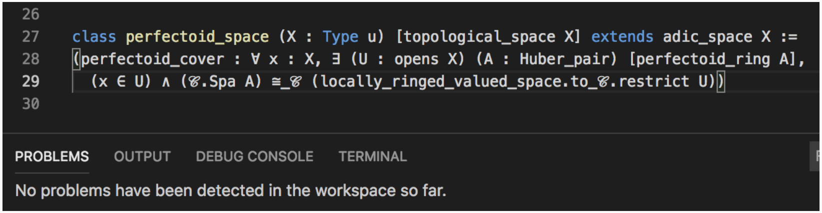
Peter Scholze won a Fields medal in 2018 for “transforming arithmetic algebraic geometry over p-adic fields through his introduction of perfectoid spaces, with application to Galois representations, and for the development of new cohomology theories.” Within the past 10 years in number theory, this one definition above all others has revolutionised the way (some) people think. The definition is highly nontrivial, an unusual geometric object created from an extremely non-Noetherian ring.

Perfectoids

Peter Scholze won a Fields medal in 2018 for "transforming arithmetic algebraic geometry over p-adic fields through his introduction of perfectoid spaces, with application to Galois representations, and for the development of new cohomology theories." Within the past 10 years in number theory, this one definition above all others has revolutionised the way (some) people think. The definition is highly nontrivial, an unusual geometric object created from an extremely non-Noetherian ring.

Last week Kevin Buzzard, Johan Commelin, Patrick Massot (and others) completed a long term project to define a perfectoid space formally in Lean.

Perfectoids

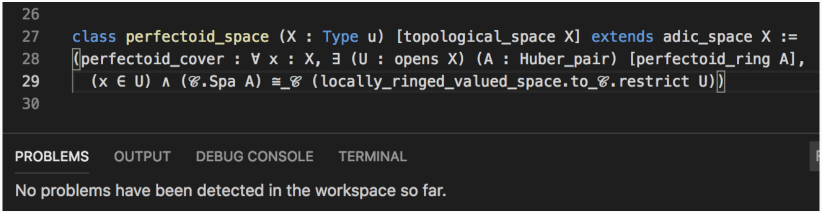


```
class perfectoid_space (X : Type u) [topological_space X] extends adic_space X :=
(perfectoid_cover : ∀ x : X, ∃ (U : opens X) (A : Huber_pair) [perfectoid_ring A],
  (x ∈ U) ∧ (𝒞.Spa A) ≅_𝒞 (locally_ringed_valued_space.to_𝒞.restrict U))
```

```
No problems have been detected in the workspace so far.
```

Lean has accepted the chain of definitions that lead to this are all valid, topological spaces, sheaves, valuations, adic spaces, perfectoid rings,...

Perfectoids



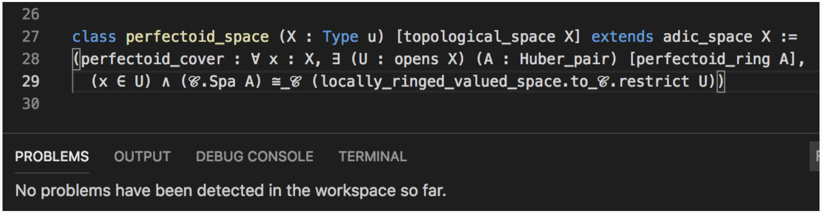
```
class perfectoid_space (X : Type u) [topological_space X] extends adic_space X :=
(perfectoid_cover : ∀ x : X, ∃ (U : opens X) (A : Huber_pair) [perfectoid_ring A],
  (x ∈ U) ∧ (𝒞.Spa A) ≅_𝒞 (locally_ringed_valued_space.to_𝒞.restrict U))
```

No problems have been detected in the workspace so far.

Lean has accepted the chain of definitions that lead to this are all valid, topological spaces, sheaves, valuations, adic spaces, perfectoid rings,...

It is difficult to estimate the amount of human effort expended to achieve this.

Perfectoids



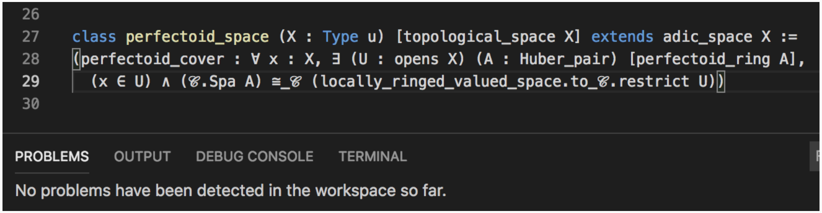
```
class perfectoid_space (X : Type u) [topological_space X] extends adic_space X :=
(perfectoid_cover : ∀ x : X, ∃ (U : opens X) (A : Huber_pair) [perfectoid_ring A],
  (x ∈ U) ∧ (𝒞.Spa A) ≅_𝒞 (locally_ringed_valued_space.to_𝒞.restrict U))
```

No problems have been detected in the workspace so far.

Lean has accepted the chain of definitions that lead to this are all valid, topological spaces, sheaves, valuations, adic spaces, perfectoid rings,...

It is difficult to estimate the amount of human effort expended to achieve this. Their work relies on that of many others who are building `mathlib`, a general purpose library of mathematics from the ground up.

Perfectoids



```
class perfectoid_space (X : Type u) [topological_space X] extends adic_space X :=
(perfectoid_cover : ∀ x : X, ∃ (U : opens X) (A : Huber_pair) [perfectoid_ring A],
  (x ∈ U) ∧ (𝒞.Spa A) ≅_𝒞 (locally_ringed_valued_space.to_𝒞.restrict U))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

No problems have been detected in the workspace so far.

Lean has accepted the chain of definitions that lead to this are all valid, topological spaces, sheaves, valuations, adic spaces, perfectoid rings,...

It is difficult to estimate the amount of human effort expended to achieve this. Their work relies on that of many others who are building `mathlib`, a general purpose library of mathematics from the ground up.

However, I would guess it compares favourably to the length of time needed to teach a human with zero mathematical background

A call to arms

I want to learn more about this!

A call to arms

I want to learn more about this!

Learning is better with others, the online lean community is great, (I posted and asked them to roast me and all they gave me was helpful tips), but nothing beats in person discussion.

A call to arms

I want to learn more about this!

Learning is better with others, the online lean community is great, (I posted and asked them to roast me and all they gave me was helpful tips), but nothing beats in person discussion.

If you think this sounds fun and/or interesting let me know, I'd love to organise a casual semi-regular meetup, to play with this together and collaborate!