Computers and Theorems
- an introduction to proof assistance and verification

Veronika Ertl

Fakultät für Mathematik
Universität Regensburg

HIOB-Seminar

1st February 2021

What are proof assistants?

Formal verification...

...involves the use of logical and computational methods to establish claims that are expressed in precise mathematical terms.

- ordinary mathematical theorems
- claims that pieces of hardware or software, network protocols, and mechanical and hybrid systems meet their specifications

What are proof assistants?

Formal verification...

...involves the use of logical and computational methods to establish claims that are expressed in precise mathematical terms.

- ordinary mathematical theorems
- claims that pieces of hardware or software, network protocols, and mechanical and hybrid systems meet their specifications

In practice...

... there is not a sharp distinction between verifying a piece of mathematics and verifying the correctness of a system!

Reason: Formal verification requires to describe such systems in mathematical terms. Then establishing claims concerning their correctness is a form of theorem proving.

To support a mathematical claim,...

... one has to provide a proof. Most conventional proof methods can be reduced to a set of axioms and rules in some foundational system.

With this reduction, a computer can help in two ways:

- It can help find a proof.
- It can help verify that a proof is correct.

In practice...

... there is not a sharp distinction between verifying a piece of mathematics and verifying the correctness of a system!

Reason: Formal verification requires to describe such systems in mathematical terms. Then establishing claims concerning their correctness is a form of theorem proving.

To support a mathematical claim,...

... one has to provide a proof. Most conventional proof methods can be reduced to a set of axioms and rules in some foundational system.

With this reduction, a computer can help in two ways:

- It can help find a proof.
- It can help verify that a proof is correct.

In practice...

... there is not a sharp distinction between verifying a piece of mathematics and verifying the correctness of a system!

Reason: Formal verification requires to describe such systems in mathematical terms. Then establishing claims concerning their correctness is a form of theorem proving.

To support a mathematical claim,...

... one has to provide a proof. Most conventional proof methods can be reduced to a set of axioms and rules in some foundational system.

With this reduction, a computer can help in two ways:

- It can help find a proof.
- It can help verify that a proof is correct.

Automated reasoning systems...

... focus on the finding aspect. They strive for power and efficiency, often at the expense of guaranteed soundness.

Interactive reasoning systems...

... focus on the verification aspect. They require that every claim is supporting by a proof in a suitable axiomatic foundation.

- The most widely used proof assistants today try to bridge the gap between automated and interactive theorem proving.
- The goal is to support both mathematical reasoning and reasoning about complex systems, and to verify claims in both domains.

One such theorem prover, which is supported by well-known mathematicians is LEAN.

Automated reasoning systems...

... focus on the finding aspect. They strive for power and efficiency, often at the expense of guaranteed soundness.

Interactive reasoning systems...

... focus on the verification aspect. They require that every claim is supporting by a proof in a suitable axiomatic foundation.

- The most widely used proof assistants today try to bridge the gap between automated and interactive theorem proving.
- The goal is to support both mathematical reasoning and reasoning about complex systems, and to verify claims in both domains.

One such theorem prover, which is supported by well-known mathematicians is LEAN.

Automated reasoning systems...

... focus on the finding aspect. They strive for power and efficiency, often at the expense of guaranteed soundness.

Interactive reasoning systems...

... focus on the verification aspect. They require that every claim is supporting by a proof in a suitable axiomatic foundation.

- The most widely used proof assistants today try to bridge the gap between automated and interactive theorem proving.
- The goal is to support both mathematical reasoning and reasoning about complex systems, and to verify claims in both domains.

One such theorem prover, which is supported by well-known mathematicians is LEAN.

Automated reasoning systems...

... focus on the finding aspect. They strive for power and efficiency, often at the expense of guaranteed soundness.

Interactive reasoning systems...

... focus on the verification aspect. They require that every claim is supporting by a proof in a suitable axiomatic foundation.

- The most widely used proof assistants today try to bridge the gap between automated and interactive theorem proving.
- The goal is to support both mathematical reasoning and reasoning about complex systems, and to verify claims in both domains.

One such theorem prover, which is supported by well-known mathematicians is LEAN.

Why use a proof assistant/theorem prover?

Currently the available computer proof systems are not good enough to tell us anything new relevant for mathematical research.

Using a theorem prover...

... involves *digitising mathematics*. And history seems to show that digitising anything enables us to do new things.

The more people are familiar with the software the earlier interesting things might happen!

Why use a proof assistant/theorem prover?

Currently the available computer proof systems are not good enough to tell us anything new relevant for mathematical research.

Using a theorem prover...

... involves *digitising mathematics*. And history seems to show that digitising anything enables us to do new things.

The more people are familiar with the software the earlier interesting things might happen!

In the future proof assistants might help us in

- Teaching (Verified course notes, data base for students and lecturers, tools which attempt example sheet questions by applying theorems from the course notes ...)
- Interaction/collaboration (Computer scientists will begin to understand what math happens in math departments.)
- Research (filling in proof of lemmas, offering search tools for theorems,...)
- Avoid mathematical mistakes.

Until then, there is a lot of work to do!

Supporters of computer proof verification among mathematicians include/included:

- Vladimir Voevodksy (IAS, 1966-2017)
- Kevin Buzzard (Imperial College)
- Johan Comelin (Freiburg)
- Patric Massot (Paris-Sud)
- ...

Some mathematics that has been formalised in Lean

- Schemes
- Witt vectors
- Perfectoid spaces
- Hensel's lemma over the p-adic integers
- ...

Supporters of computer proof verification among mathematicians include/included:

- Vladimir Voevodksy (IAS, 1966-2017)
- Kevin Buzzard (Imperial College)
- Johan Comelin (Freiburg)
- Patric Massot (Paris-Sud)
- ...

Some mathematics that has been formalised in Lean

- Schemes
- Witt vectors
- Perfectoid spaces
- Hensel's lemma over the p-adic integers
- ...

Developing a proof assistant

This involves two basic steps:

- foundational work: find the best foundational theory to formalise mathematics
  - type theory/set theory (Metamath, Isabelle,...)
  - dependent type theory/homotopy type theory (Coq, Lean,...)
- coverage work: tries to formalise as much of existing mathematics as possible
  - UniMath project (Voevodsky), uses Coq
  - Xena project (Buzzard), uses Lean
  - ...

Developing a proof assistant

This involves two basic steps:

- foundational work: find the best foundational theory to formalise mathematics
  - type theory/set theory (Metamath, Isabelle,...)
  - dependent type theory/homotopy type theory (Coq, Lean,...)
- coverage work: tries to formalise as much of existing mathematics as possible
  - UniMath project (Voevodsky), uses Coq
  - Xena project (Buzzard), uses Lean
  - ...

Developing a proof assistant

This involves two basic steps:

- foundational work: find the best foundational theory to formalise mathematics
  - type theory/set theory (Metamath, Isabelle,...)
  - dependent type theory/homotopy type theory (Coq, Lean,...)
- coverage work: tries to formalise as much of existing mathematics as possible
  - UniMath project (Voevodsky), uses Coq
  - Xena project (Buzzard), uses Lean
  - ...

A first example

Example

Let A, B be some statements or "propositions", then $A \wedge B \Rightarrow B \wedge A$.

Proof.

If we have a proof for $A \wedge B$, then we have a proof for B (*right and-elimination*).
If we have a proof for $A \wedge B$, then we have a proof for A (*left and-elimination*).
Thus, if we have a proof for $A \wedge B$, then we have a proof for B and A.
But then we have a proof for $B \wedge A$ (*and-introduction*). $\square$

In symbolic logic:

$$\frac{\dfrac{A \wedge B}{B} \qquad \dfrac{A \wedge B}{A}}{B \wedge A}$$

A first example

Example

Let A, B be some statements or "propositions", then $A \wedge B \Rightarrow B \wedge A$.

Proof.

If we have a proof for $A \wedge B$, then we have a proof for B (*right and-elimination*).
If we have a proof for $A \wedge B$, then we have a proof for A (*left and-elimination*).
Thus, if we have a proof for $A \wedge B$, then we have a proof for B and A.
But then we have a proof for $B \wedge A$ (*and-introduction*). $\square$

In symbolic logic:

$$\frac{\dfrac{A \wedge B}{B} \qquad \dfrac{A \wedge B}{A}}{B \wedge A}$$

A first example

Example

Let A, B be some statements or "propositions", then $A \wedge B \Rightarrow B \wedge A$.

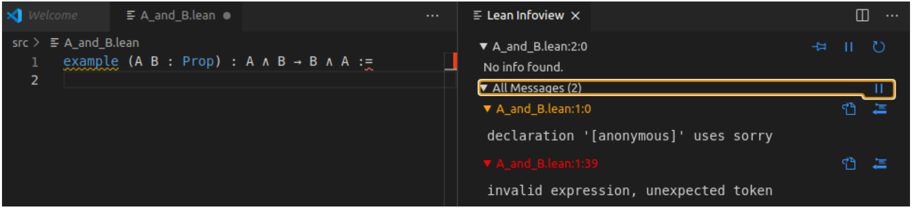
Proof.

If we have a proof for $A \wedge B$, then we have a proof for B (*right and-elimination*).
If we have a proof for $A \wedge B$, then we have a proof for A (*left and-elimination*).
Thus, if we have a proof for $A \wedge B$, then we have a proof for B and A.
But then we have a proof for $B \wedge A$ (*and-introduction*). $\square$

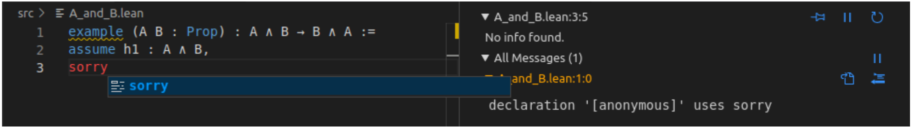
In symbolic logic:

$$\frac{\dfrac{A \wedge B}{B} \qquad \dfrac{A \wedge B}{A}}{B \wedge A}$$

In Lean:


Welcome
A_and_B.lean
Lean Infoview
src > A_and_B.lean
example (A B : Prop) : A ∧ B → B ∧ A :=
A_and_B.lean:2:0
No info found.
All Messages (2)
A_and_B.lean:1:0
declaration '[anonymous]' uses sorry
A_and_B.lean:1:39
invalid expression, unexpected token


src > A_and_B.lean
example (A B : Prop) : A ∧ B → B ∧ A :=
sorry
A_and_B.lean:2:5
No info found.
All Messages (1)
A_and_B.lean:1:0
declaration '[anonymous]' uses sorry


src > A_and_B.lean
example (A B : Prop) : A ∧ B → B ∧ A :=
assume h1 : A ∧ B,
sorry
sorry
A_and_B.lean:3:5
No info found.
All Messages (1)
and_B.lean:1:0
declaration '[anonymous]' uses sorry

src > A_and_B.lean

```
example (A B : Prop) : A ∧ B → B ∧ A :=
assume h1 : A ∧ B,
have h2 : A, from and.left h1,
sorry
```

▼ A_and_B.lean:3:30

No info found.

▼ All Messages (1)

▼ A_and_B.lean:1:0

```
declaration '[anonymous]' uses sorry
```

src > A_and_B.lean

```
example (A B : Prop) : A ∧ B → B ∧ A :=
assume h1 : A ∧ B,
have h2 : A, from and.left h1,
have h3 : B, from and.right h1,
sorry
```

▼ A_and_B.lean:4:31

No info found.

▼ All Messages (1)

▼ A_and_B.lean:1:0

```
declaration '[anonymous]' uses sorry
```

src > A_and_B.lean

```
example (A B : Prop) : A ∧ B → B ∧ A :=
assume h1 : A ∧ B,
have h2 : A, from and.left h1,
have h3 : B, from and.right h1,
show B ∧ A, from and.intro h3 h2
```

▼ A_and_B.lean:5:32

No info found.

▼ All Messages (0)

No messages.

Lets do something a little bit more exciting

Theorem

Let A and B be some propositions. Then $\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$.

This will be a proof by contradiction.
Note that $\neg A$ is the same as "$A \Rightarrow$ false ".

Proof (Step 1).

We always assume we have a proof for $\neg(A \wedge B)$.
In a first step, we assume A is true, and from this conclude $\neg B$:
We assume B, thus, we know $A \wedge B$. Applying our main hypothesis, we obtain "false". Thus $\neg B$. But then we also know $\neg A \vee \neg B$ (right or-introduction). □

Lets do something a little bit more exciting

Theorem

Let A and B be some propositions. Then $\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$*.*

This will be a proof by contradiction.
Note that $\neg A$ is the same as "$A \Rightarrow$ false ".

Proof (Step 1).

We always assume we have a proof for $\neg(A \wedge B)$.
In a first step, we assume A is true, and from this conclude $\neg B$:
We assume B, thus, we know $A \wedge B$. Applying our main hypothesis, we obtain "false". Thus $\neg B$. But then we also know $\neg A \vee \neg B$ (right or-introduction). □

Lets do something a little bit more exciting

Theorem

Let A and B be some propositions. Then $\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$.

This will be a proof by contradiction.
Note that $\neg A$ is the same as "$A \Rightarrow$ false ".

Proof (Step 1).

We always assume we have a proof for $\neg(A \wedge B)$.
In a first step, we assume A is true, and from this conclude $\neg B$:
We assume B, thus, we know $A \wedge B$. Applying our main hypothesis, we obtain "false". Thus $\neg B$. But then we also know $\neg A \vee \neg B$ (right or-introduction). $\square$

Proof (Step 2).

In a next step, we assume again the main hypothesis $\neg(A \wedge B)$ and in addition $\neg(\neg A \vee \neg B)$ and show “false” from this. This will be the core of our contradiction argument in the third step.
We show first $\neg A$ from the two hypothesis:
Assume A. Applying Step 1 to the main hypothesis and the assumption of A, we obtain $\neg A \vee \neg B$. But applying the hypothesis $\neg(\neg A \vee \neg B)$, we obtain “false”, and thus, $\neg A$.
Thus we also have $\neg A \vee \neg B$ from the same hypothesis (left or-introduction). But applying $\neg(\neg A \vee \neg B)$ we conclude “false”. $\square$

Proof (Step 3).

Lastly we just assume the main hypothesis, that is, we have a proof for $\neg(A \wedge B)$. We show $\neg A \vee \neg B$ by contradiction.
Thus assume $\neg(\neg A \vee \neg B)$. But now we can apply Step 2 to the main hypothesis and $\neg(\neg A \vee \neg B)$, and obtain a contradiction. Thus we conclude $\neg A \vee \neg B$. $\square$

Proof (Step 2).

In a next step, we assume again the main hypothesis $\neg(A \wedge B)$ and in addition $\neg(\neg A \vee \neg B)$ and show "false" from this. This will be the core of our contradiction argument in the third step.

We show first $\neg A$ from the two hypothesis:

Assume A. Applying Step 1 to the main hypothesis and the assumption of A, we obtain $\neg A \vee \neg B$. But applying the hypothesis $\neg(\neg A \vee \neg B)$, we obtain "false", and thus, $\neg A$.

Thus we also have $\neg A \vee \neg B$ from the same hypothesis (left or-introduction). But applying $\neg(\neg A \vee \neg B)$ we conclude "false". $\square$

Proof (Step 3).

Lastly we just assume the main hypothesis, that is, we have a proof for $\neg(A \wedge B)$. We show $\neg A \vee \neg B$ by contradiction.

Thus assume $\neg(\neg A \vee \neg B)$. But now we can apply Step 2 to the main hypothesis and $\neg(\neg A \vee \neg B)$, and obtain a contradiction. Thus we conclude $\neg A \vee \neg B$. $\square$

src > not_A_and_B.lean

```lean
open classical
variables {A B C : Prop}

-- Prove ¬ (A ∧ B) → ¬ A ∨ ¬ B by replacing the sorry's below
-- by proofs.

lemma step1 (h1 : ¬ (A ∧ B)) (h2 : A) : ¬ A ∨ ¬ B :=
have ¬ B, from
  assume : B,
  have h3 : A ∧ B, from and.intro h2 this,
  show false, from h1 h3,
show ¬ A ∨ ¬ B, from or.inr this

lemma step2 (h1 : ¬ (A ∧ B)) (h2 : ¬ (¬ A ∨ ¬ B)) : false :=
have  ¬ A, from
  assume : A,
  have ¬ A ∨ ¬ B, from step1 h1 ‹A›,
  show false, from h2 this,
show false, from h2 (or.inl this)

theorem step3 (h : ¬ (A ∧ B)) : ¬ A ∨ ¬ B :=
by_contradiction
  (assume h' : ¬ (¬ A ∨ ¬ B),
    show false, from step2 h h')
```

Using the library

Lean has several libraries that can be used to build new proofs - mathematical and code libraries. The next example is to demonstrate how to make use of these libraries.

Euclid's theorem

In the natural numbers, there are infinitely many prime numbers.

Proof.

Let $n \in \mathbb{N}$. Its factorial $n!$ is divisible by all natural numbers between 2 and n. Hence $n! + 1$ is not divisible by any of the natural numbers between 2 and n. Thus $n! + 1$ is either prime or divisible by a prime larger than n. Either way, for every natural number n, there is a prime p bigger than n. Consequently, there are infinitely many primes. □

Using the library

Lean has several libraries that can be used to build new proofs - mathematical and code libraries. The next example is to demonstrate how to make use of these libraries.

Euclid's theorem

In the natural numbers, there are infinitely many prime numbers.

Proof.

Let $n \in \mathbb{N}$. Its factorial $n!$ is divisible by all natural numbers between 2 and n. Hence $n! + 1$ is not divisible by any of the natural numbers between 2 and n. Thus $n! + 1$ is either prime or divisible by a prime larger than n. Either way, for every natural number n, there is a prime p bigger than n. Consequently, there are infinitely many primes. □

```
open nat

theorem infinitude_of_primes : ∀ N, ∃ p ≥ N, prime p :=

begin
  intro N,

  let M := factorial N + 1,
  let p := min_fac M,

  have hp : prime p :=
  begin
    refine min_fac_prime _,
    have : factorial N > 0 := factorial_pos N,
    linarith,
  end,


  use p,
  split,
  { by_contradiction,
    have h₁ : p ∣ factorial N + 1 := by  exact min_fac_dvd M,
    have h₂ : p ∣ factorial N :=
    begin
      refine hp.dvd_factorial.mpr _,
      exact le_of_not_ge h,
    end,
    have h : p ∣ 1 := (nat.dvd_add_right h₂).mp h₁,
    exact prime.not_dvd_one hp h},
  {exact hp,},
end
```

Thank you for your attention!