Logical reasoning and programming

Lecture 8: Proof assistants

Karel Chvalovský

CIIRC CTU

November 10, 2025

What do we have so far?

We have introduced diverse techniques that make it possible to automatically solve problems in logics of various strengths:

- propositional logic,
- (ground) first-order logic with theories (SMT), and
- first-order logic.

However, if we want to solve real problems, we usually need to combine many small results together and hence we need a tool that makes this process as convenient as possible. These tools are called proof assistants (or interactive theorem provers).

What do we want from proof assistants?

The basic things that a reasonable proof assistant should provide are, for example:

- appropriate formal language,
- proof checking,
- assistance with proving,
  - current proof state (what remains to be proved),
  - library search,
  - automatic proof search,
- nice presentation.

However, we have to start with a formalization of our problem.

formal language $\Rightarrow$ formal specification $\Rightarrow$ formal proof

Formalization

We formulate our problem in terms of axioms, definitions, lemmata, and theorems. We prove things using allowed inference rules. The goal is to certify that all proofs are valid.

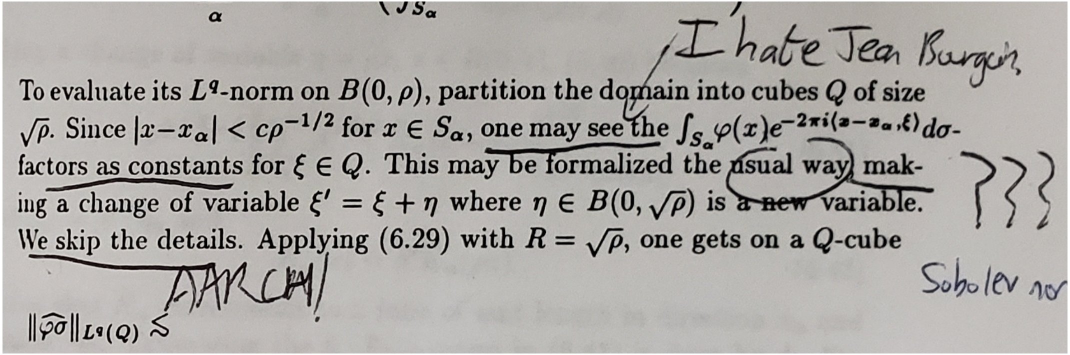
It is usually not so difficult to check a validity of a proof, however, it is surprisingly difficult to write down all the details necessary to make it possible.

An obvious candidate for a straightforward formalization is mathematics, which is already quite formal. However, even in mathematics a full formalization is far from easy.

Why is it so difficult even in mathematics?

We require the precise representation of

- objects
  - how should we represent real numbers, division, computation, graphs, ...
  - how to switch between different representations
- proof steps
  - "use the method introduced in the above paragraph"
  - "the rest is a standard diagonalization argument"
  - "the following reasoning holds up to a set of measure zero"



To evaluate its L^q-norm on $B(0,\rho)$, partition the domain into cubes Q of size $\sqrt{\rho}$. Since $|x-x_\alpha| < c\rho^{-1/2}$ for $x \in S_\alpha$, one may see the $\int_{S_\alpha}\varphi(x)e^{-2\pi i(x-x_\alpha,\xi)}d\sigma$-factors as constants for $\xi \in Q$. This may be formalized the usual way making a change of variable $\xi' = \xi + \eta$ where $\eta \in B(0,\sqrt{\rho})$ is a new variable. We skip the details. Applying (6.29) with $R = \sqrt{\rho}$, one gets on a Q-cube

$\|\widehat{\varphi\sigma}\|_{L^q(Q)} \lesssim$

source: Terry Tao about Jean Bourgain's paper, see blog

Practical notes about formalization of mathematics

- It is necessary to write down all the details, people do not do that for efficiency and clarity reasons (and sometimes we make mistakes due to that).
- Some notions are hard to formalize in some systems.
- de Bruijn factor — the ratio of lengths of a formal proof to the corresponding informal proof (LaTeX source) is surprisingly stable ($\sim$4). Apparently people tend to formalize problems where the overhead is not so big. For more details, see Wiedijk's page on the de Bruijn factor (from 2012).
- 1 week $\approx$ 1 page of a math textbook ($\approx \frac{1}{2}$ day of classical writing).

Mistakes in mathematical proofs

Vladimir Voevodsky (1966–2017) was a very famous mathematician, a professor at the Institute for Advanced Studies and a Fields medalist, who later in his career became interested in formal methods.

> *In 1999–2000 . . . [I discovered] that the proof of a key lemma in my paper contained a mistake and that the lemma, as stated, could not be salvaged. . . . This story got me scared. Starting from 1993, multiple groups of mathematicians studied my 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.*
>
> *Voevodsky, The Institute Letter Summer 2014*

Principia Mathematica

It was a fully formal mathematical text (in 3 volumes, the first edition was published in 1910, 1912, and 1913) by Whitehead and Russell. It was simultaneously a huge success and failure. For example, it takes over 300 pages to prove $1 + 1 = 2$.

$$*54\cdot43.\quad \vdash :. \alpha, \beta \in 1 . \supset : \alpha \cap \beta = \Lambda . \equiv . \alpha \cup \beta \in 2$$

Dem.

$$\vdash . *54\cdot26 . \supset \vdash :. \alpha = \iota`x . \beta = \iota`y . \supset : \alpha \cup \beta \in 2 . \equiv . x \neq y .$$

$$[*51\cdot231] \qquad \equiv . \iota`x \cap \iota`y = \Lambda .$$

$$[*13\cdot12] \qquad \equiv . \alpha \cap \beta = \Lambda \qquad (1)$$

$$\vdash . (1) . *11\cdot11\cdot35 . \supset$$

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

$$\vdash . (2) . *11\cdot54 . *52\cdot1 . \supset \vdash . \mathrm{Prop}$$

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

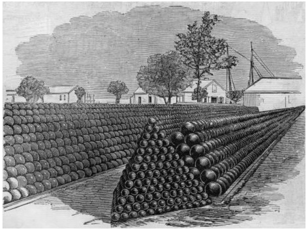
source: wiki

Russell during the work on it arguably contemplated suicide and said: "my intellect never quite recovered from the strain . . . I have been ever since definitely less capable of dealing with difficult abstractions than I was before."

Kepler's conjecture

Problem (Sir Walter Raleigh)

Determine formulae for the number of cannonballs in regularly stacked piles.

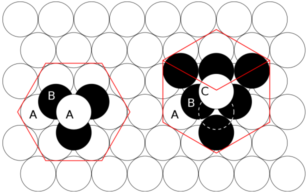


Claim (Thomas Harriot, ∼1587)

The density of the faced-centered cubic packing is $\frac{\pi}{\sqrt{18}} \approx 0.74048.$

Conjecture (Johannes Kepler, 1611)

No packing of balls of the same radius in three dimensions has density greater than the face-centered cubic packing.



Tom Hales's solution

Theorem (Tom Hales, 1998)

No packing of balls of the same radius in three dimensions has density greater than the face-centered cubic packing.

Paper was submitted to *Annals of Mathematics*

- received September 4, 1998
  - 250 pages, source codes + data (graph enumeration, non-linear optimization, and linear programming)
  - 12 reviewers gave up after 4 years—"They have not been able to certify the correctness of the proof, and will not be able to certify it in the future, because they have run out of energy to devote to the problem."
- a shorten version (123 pages) was eventually published (August 16, 2005) and the *Annals* are checking computer programs only for obvious errors since then

FlysPecK

Formal Proof of the Kepler conjecture, flyspeck = examine closely

As a result Hales initiated a project to completely formalize the proof. It was completed on August 10, 2014. For details see GitHub.

- the Flyspeck book (Dense Sphere Packing) was created to make the formalization process easier (informal text)
- formalized in HOL Light (formal text) and Isabelle
- 20,000 lemmata in geometry, analysis, and graph theory
- three separate computational sub-claims (the most difficult one took 5,000 CPU hours to verify)

Verifying computations

We can verify a computation many ways, for example,

- the computation constructs a proof as a by-product (Flyspeck: non-linear bounds),
- verify certificates (Flyspeck: proof sketches),
- verify the algorithm, then execute it with a trusted evaluator (four color theorem),
- verify the algorithm, extract code, and run it trusting a compiler or interpreter (Flyspeck: enumeration of tame graphs).

Liquid Tensor Experiment

In December 2020, Peter Scholze posted a challenge, see here. There are many mathematical reasons why it is interesting and

> *I spent much of 2019 obsessed with the proof of this theorem, almost getting crazy over it. In the end, we were able to get an argument pinned down on paper, but I think nobody else has dared to look at the details of this, and so I still have some small lingering doubts.*
>
> *I have occasionally been able to be very persuasive even with wrong arguments. (Fun fact: In the selection exams for the international math olympiad, twice I got full points for a wrong solution. Later, I once had a full proof of the weight-monodromy conjecture that passed the judgment of some top mathematicians, but then it turned out to contain a fatal mistake.)*
>
> *I think this may be my most important theorem to date.*

It was completed in July 2022, see here. Terry Tao also formalized a problem in Lean, see here, and works on a new one, see here.

Proof assistants and LLMs

There are many papers combining proof assistants, mostly Lean, and LLMs. For example:

- There have been recent successes in solving problems from the International Mathematical Olympiad (IMO).
- A solution (arXiv:2510.19804) to an Erdős problem that was "vibe coded" in Lean, see here for details. (this is not the infamous recent OpenAI story)

Forbidden Sidon subsets of perfect difference sets, featuring a human-assisted proof

Boris Alexeev   ChatGPT*   Lean†   Dustin G. Mixon‡§

Abstract

We resolve a $1000 Erdős prize problem, complete with formal verification generated by a large language model.

source: here

Pentium FDIV bug

It was a bug in the floating point unit that affected early Intel Pentium processors and costed Intel \$475 M. The reason was that 5 cells were missing in a programmable logic array.

An elegant demonstration of the problem is that

$$4195835/3145727$$

should return

$$1.333820\ldots$$

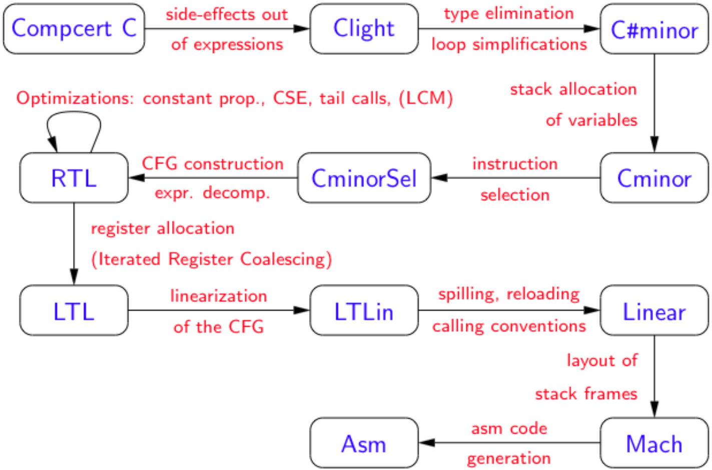
but affected processors returned

$$1.333739\ldots$$

As a consequence, Intel improved investments in formal verification efforts. For example, various floating-point algorithms were formally verified in HOL Light. Maybe surprisingly, it requires a non-trivial mathematics.

CompCert

It is a formally verified (in Rocq) optimizing compiler for a large fragment of C language for PowerPC, ARM, RISC-V and x86 (32 and 64 bits) architectures. Hence all the proved properties on the source code hold also for the generated executable.



For more details see their webpage, the image is from there.

seL4

A verified (in Isabelle) L4 microkernel, but only the ARM, ARM_HYP, RISC-V, and x86 versions have a full code-level functional correctness proof; the implementation (written in C) adheres to its specification.

We have some more properties (for ARM and RISC-V):

- the binary code which executes on the hardware is a correct translation of the C code and hence there is no need to trust the compiler,
- a correct usage of the seL4 specification also ensures integrity and confidentiality (security properties), and
- provable upper bound latencies on kernel operations.

For more details about seL4 see their webpage, or presentations from the recent seL4 Summit 2025 that was in September in Prague.

What the seL4 proofs assume

Assembly code, boot code, machine interface, hardware, and DMA. See here.

Mathematics vs. computer science

Clearly, formal verification problems in mathematics and computer science are usually different:

	math	cs
size of statements	small	large
formulation of statements	easy to get right	easy to get wrong
proofs	intricate	straightforward
	interesting	boring

However, there is no sharp line between them, because even problems in computer science are expressed using mathematical terms and usually depend on mathematical results.

These days basically all major software and hardware companies pay close attention to formal verification.

Not surprisingly, it is much easier to verify software that was created with the verification process in mind, see Dafny.

Formal methods in Amazon

Amazon experimented with TLA (Temporal Logic of Actions):

> *In industry, formal methods have a reputation for requiring a huge amount of training and effort to verify a tiny piece of relatively straightforward code, so the return on investment is justified only in safety-critical domains (such as medical systems and avionics). Our experience with TLA+ shows this perception to be wrong. At the time of this writing, Amazon engineers have used TLA+ on 10 large complex real-world systems. In each, TLA+ has added significant value, either finding subtle bugs we are sure we would not have found by other means, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness.*
>
> *We also initially avoid mentioning what TLA stands for, as doing so would give an incorrect impression of complexity.*

(Newcombe et al. 2015)

Software correctness in Amazon (2023)

> ***Cook***: *. . . The storage team, for example, is able to be much more agile and be much more aggressive in the programs that they write because of the formal methods. They're able to write code that otherwise they might not want to deploy because they wouldn't be as confident about it, and they're deploying four times as fast. . . .*
>
> ***Kröning***: *There are actually a good number of stories wherein engineering teams didn't dare to roll out a particular feature or design revision or design variant that offers clear benefits—like being faster, using less power—because they just couldn't gain the confidence that it's actually right under all circumstances.*

source: Automated reasoning at Amazon: A conversation

Another AWS example: s2n-bignum is a collection of integer arithmetic routines designed for cryptographic applications; primary goals are performance and assurance.

Formalizing 100 Theorems

A list of problems containing 100 “interesting” theorems; many of them are elementary.

System	#theorems
Isabelle	92
HOL Light	89
Lean	82
Rocq	79
Metamath	74
Mizar	71
all together	99

source: Formalizing 100 Theorems by Wiedijk

The missing theorem is Fermat’s Last Theorem (an ongoing effort in Lean).

Note that, for example, Rocq (formerly Coq) has some big formalizations like the four color problem or Feit–Thompson (odd order) theorem (not on the list, but 4,000 definitions and 13,000 theorems based on 250 pages of text from 2 books).

Proof languages

Similarly to programming languages, there are various types of proof languages.
Moreover, they use different foundations.

Procedural style

A very common approach, because a formal proof usually says how it is proved.
However, it is far less readable and very language specific.

Declarative style

It says what is to be proved. It is much easier to use external tools, modify proofs etc.
It is usually more verbose and harder to script.

This style was pioneered by Mizar.

$\sqrt{2}$ is irrational

It is the first problem on the list. For a formalization of this problem in many proof assistants see The Seventeen Provers of the World by Wiedijk; the following text is from it.

THEOREM 43 (PYTHAGORAS' THEOREM). *$\sqrt{2}$ is irrational.*
The traditional proof ascribed to Pythagoras runs as follows. If $\sqrt{2}$ is rational, then the equation

$$a^2 = 2b^2 \tag{4.3.1}$$

is soluble in integers a, b with $(a, b) = 1$. Hence a^2 is even, and therefore a is even. If $a = 2c$, then $4c^2 = 2b^2$, $2c^2 = b^2$, and b is also even, contrary to the hypothesis that $(a, b) = 1$. □

$\sqrt{2}$ is irrational in Isabelle

```
theorem sqrt_prime_irrational:
  fixes p :: nat
  assumes "prime p"
  shows "sqrt p ∉ ℚ"
proof
  from ‹prime p› have p: "p > 1" by (rule prime_gt_1_nat)
  assume "sqrt p ∈ ℚ"
  then obtain m n :: nat
    where n: "n ≠ 0"
      and sqrt_rat: "¦sqrt p¦ = m / n"
      and "coprime m n" by (rule Rats_abs_nat_div_natE)
  have eq: "m² = p * n²"
  proof -
    from n and sqrt_rat have "m = ¦sqrt p¦ * n" by simp
    then have "m² = (sqrt p)² * n²" by (simp add: power_mult_distrib)
    also have "(sqrt p)² = p" by simp
    also have "... * n² = p * n²" by simp
    finally show ?thesis by linarith
  qed
  have "p dvd m ∧ p dvd n"
  proof
    from eq have "p dvd m²" ..
    with ‹prime p› show "p dvd m" by (rule prime_dvd_power)
    then obtain k where "m = p * k" ..
    with eq have "p * n² = p² * k²" by algebra
    with p have "n² = p * k²" by (simp add: power2_eq_square)
    then have "p dvd n²" ..
    with ‹prime p› show "p dvd n" by (rule prime_dvd_power)
  qed
  then have "p dvd gcd m n" by simp
  with ‹coprime m n› have "p = 1" by simp
  with p show False by simp
qed

corollary sqrt_2_not_rat: "sqrt 2 ∉ ℚ"
  using sqrt_prime_irrational [of 2] by simp
```

source: Isabelle

$\sqrt{2}$ is irrational in Metamath

A compressed version is

```
${
  $d x y $.
  $( The square root of 2 is irrational. $)
  sqr2irr $p |- ( sqr ‘ 2 ) e/ QQ $=
    ( vx vy c2 csqr cfv cq wnel wcel wn cv cdiv co wceq cn wrex cz cexp
    cmulc sqr2irrlem3 sqr2irrlem5 bi2rexa mtbir cc0 clt wbr wa wi wb nngt0t
    adantr cr ax0re ltmuldivt mp3an1 nnret zret syl2an mpd ancoms 2re 2pos
    sqrgt0i breq2 mpbii syl5bir cc nncnt mulzer2t syl breq1d adantl sylibd
    exp r19.23adv anc2li elnnz syl6ibr impac r19.22i2 mto elq df-nel mpbir )
    CDEZFGWDFHZIWEWDAJZBJZKLZMZBNOZAPOZWKWJANOZWLWFCQLCWGCQLRLMZBNOANOABSWIWM
    ABNNWFWGTUAUBWJWJAPNWFPHZWJWFNHZWNWJWNUCWFUDUEZUFWOWNWJWPWNWIWPBNWNWGNHZW
    IWPUGWNWQUFZWIUCWGRLZWFUDUEZWPWRWTUCWHUDUEZWIWQWNWTXAUHZWQWNUFUCWGUDUEZXB
    WQXCWNWGUIUJWGUKHZWFUKHZXCXBUGZWQWNUCUKHXDXEXFULUCWGWFUMUNWGUOWFUPUQURUSW
    IUCWDUDUEXACUTVAVBWDWHUCUDVCVDVEWQWTWPUHWNWQWSUCWFUDWQWGVFHWSUCMWGVGWGVHV
    IVJVKVLVMVNVOWFVPVQVRVSVTABWDWAUBWDFWBWC $.
    $( [8-Jan-02] $)
$}
```

Note that the string expands to a sequence of steps, where A is vx, B is vy, . . .

An updated and more readable variant is here.

Foundations

It is non-trivial to select the correct foundations, we usually balance between simplicity and flexibility (expressiveness).

- set theories
  - Mizar, Metamath, Isabelle/ZF
  - the standard foundations for mathematics
  - problems
    - adding types
    - higher-order reasoning
    - computations
- type theories
  - simple type theory — HOL, Isabelle/HOL, HOL light
  - dependent type theory — Rocq (constructive), Lean, PVS (classical)
    - a type definition depends on a value
    - propositions as types (Curry–Howard correspondence)
- primitive recursive arithmetic (ACL2) — weak, a type of finitistic reasoning about natural numbers (quantifier-free)

Intuitionistic logic

Loosely speaking, it is a constructive variant of classical logic.

Brouwer–Heyting–Kolmogorov (BHK) interpretation

We say that a proof of

- $\varphi \wedge \psi$ is a proof of φ and a proof of ψ,
- $\varphi \vee \psi$ consists of selecting one of φ, ψ and a proof of the selected formula (disjunction property),
- $\varphi \to \psi$ is a transformation of a proof of φ into a proof of ψ,
- $\perp$ does not exist,
- $\neg\varphi$ is a shortcut for a proof of $\varphi \to \perp$,
- $\forall X \varphi(X)$ is a transformation of any object a into a proof of $\varphi(a)$,
- $\exists X \varphi(X)$ consists of constructing an object a (witness) and a proof of $\varphi(a)$ (existence or witness property).

Example

$\varphi \vee \neg\varphi$ has no proof, but $\varphi \to (\neg\varphi \to \psi)$ has a proof.

An intuitionistic proof

Theorem

There exist irrational numbers a and b such that a^b is rational.

Classical proof

$\sqrt{2}^{\sqrt{2}}$ is either rational, or irrational (by the law of excluded middle). If it is irrational, then $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$.

Intuitionistic proof

Use $a = \sqrt{2}$ and $b = \log_2 9$. Clearly, $a^b = 3$. It is easy to prove that b is irrational, because $\log_2 9 = \frac{m}{n}$ means $9^n = 2^m$. The constructive argument that $\sqrt{2}$ is irrational is a bit more complicated, see, e.g., here.

An intuitionistic proof

Theorem

There exist irrational numbers a and b such that a^b is rational.

Classical proof

$\sqrt{2}^{\sqrt{2}}$ is either rational, or irrational (by the law of excluded middle). If it is irrational, then $a = \sqrt{2}^{\sqrt{2}}$ and $b = \sqrt{2}$.

Intuitionistic proof

Use $a = \sqrt{2}$ and $b = \log_2 9$. Clearly, $a^b = 3$. It is easy to prove that b is irrational, because $\log_2 9 = \frac{m}{n}$ means $9^n = 2^m$. The constructive argument that $\sqrt{2}$ is irrational is a bit more complicated, see, e.g., here.

From Gelfond–Schneider theorem follows that $\sqrt{2}^{\sqrt{2}}$ is transcendental and hence irrational, see wiki.

Practical issues with proof checking

Even proof checkers contain bugs; at least one incorrect proof of a real problem has been "proved". There are two basic approaches how to deal with this problem.

de Bruijn approach

It is possible to independently check all proofs by a small program.

LCF approach

We have a small logical kernel and all complex rules can be reduced into a sequence of steps in this small kernel.

A basic observation is that the smaller amount of code, the higher the reliability.

Example

HOL Light is just 430 lines of the critical code. Rocq/Coq is historically known to be quite big.

What else can go wrong?

The most important thing is if you have a correct formulation of

- definitions and
- theorems.

This is usually easier for mathematics, because these are often given in advance.

For real-world problems, this a major issue, because

- There is a mismatch between a formal definition and real-world constraints.
- People tend to formalize problems in a way that favors the results they want.
  - For example, when people verify security models/protocols, their approach is often influenced by whether they want to show correctness, or find spurious bugs.
- Moreover, when it comes to verifying cryptography, the underlying cryptographic algorithms are usually treated as perfect.

Libraries

When we formalize a problem, it usually depends on many other results and hence we want libraries of “basic” results.

In fact, it is much more important to have many basic results than to have few big theorems. However, big results usually depend on lot of basic facts and as a by-product produce many useful basic results.

The QED manifesto was an attempt, started in 1993, to formalize all interesting mathematical knowledge and techniques. Dead since 1996, but still a source of inspiration.

Translating proofs between proof assistants

It would be nice to translate formalizations between systems, however, it is hard (sometimes even impossible thanks to different foundations) and we want “readable” outputs.

It is possible to share results between similar systems, e.g.,

- ACL2 $\rightarrow$ HOL4.

More interesting are translations like, e.g.,

- HOL Light $\rightarrow$ Isabelle/HOL,
- Isabelle/HOL $\rightarrow$ HOL Light,
- HOL Light $\rightarrow$ Coq,
- Mizar $\rightarrow$ Isabelle/ZF or Isabelle/HOL.

Note that this is similar to translating programs between different programming languages.

Automation

We can use various automation tools to simplify our task, e.g.,

- ATPs for pure logic,
- linear arithmetic decision procedures,
- Gröbner bases for solving polynomial systems, and
- counter-example finders.

Hammers

We can translate a problem into, e.g., first-order (or higher-order) logic and use an external prover (E, Vampire, Z3, CVC5, . . .) to prove it fully automatically.

Sometimes, it is non-trivial to import the external proof, but it can be used at least as a good premise selection (select relevant lemmata), which is often enough to reconstruct the proof using weaker internal provers. We usually do not trust external proofs, because they depend on the correctness of external tools, translations, . . .

Machine learning over proofs

We have various formalizations and naturally we can try to learn something on top of them. The goal is to improve automatic methods and hence make the formalization process easier.

We can learn, for example,

- relevant facts (premise selection),
- various parameters of ATPs that improve their performance,
- proof strategies,
- common proof techniques, and
- auto-completions.

Example

It is possible to prove automatically

- over 50% of Mizar and
- over 40% of Flyspeck.

Inductive proofs

We want to prove something using mathematical induction

$$P(0) \wedge \forall N(P(N) \rightarrow P(N+1)) \rightarrow \forall N(P(N)),$$

where P is a predicate over a natural number and N is a natural number. However, we claim that it holds for every predicate P; we quantify over a predicate and hence we use second-order logic. Therefore first-order theorem provers are generally not sufficient for such proofs and higher-order theorem provers, for example Satallax, are needed.

Clearly, we can use any well-founded set (no descending chains) not only natural numbers. We have inductive proofs over trees, lists, ...

Axioms vs. definitions

If we want to reason about, for example, the natural numbers, then we have two basic options:

Axioms

For example, using the Peano axioms

$$\begin{gathered}\forall X(0 \neq s(X)) \\ \forall X \forall Y(s(X) = s(Y) \to X = Y) \\ \forall X(X + 0 = X) \\ \vdots\end{gathered}$$

Definitions

For example, using the von Neumann ordinals where $0 = \emptyset$ and $s(X) = X \cup \{X\}$. Hence $1 = \{\emptyset\}, 2 = \{\emptyset, \{\emptyset\}\}, \ldots$. We can prove the Peano axioms, because we can also define $X + 0 = X$, $X \cdot 0 = 0$, $X + s(Y) = s(X + Y)$, and $X \cdot s(Y) = X \cdot Y + X$.

Axioms vs. definitions

- axioms
  - usually easier to obtain
  - sometimes unclear whether axioms are adequate for our purposes
    - an extreme case is when they are inconsistent—we can prove everything from them
- definitions
  - usually harder to obtain
  - we have a relative consistency
  - conservative extensions

Example

If we have the axiom schema of unrestricted comprehension in set theory, then we can define a set as $\{X : \varphi(X)\}$. However, for $\varphi(X)$ being $X \notin X$, we obtain Russell's paradox, because if $A = \{X : X \notin X\}$, then $A \in A$ iff $A \notin A$.

Consistency

A theory T is consistent if $T \nvdash \bot$.

We know thanks to the second Gödel incompleteness theorem that a sufficiently strong recursively enumerable theory of arithmetics cannot prove its own consistency. Note that Tarski's undefinability theorem says that, for a similarly powerful system, it is impossible to define its semantics in itself.

We have sometimes relative consistency results — if we have a theory T and produce a stronger theory $T' = T \cup \{\varphi\}$ by adding an axiom φ, then it may be possible to prove that T' is consistent, if T is consistent. We say that T' is consistent relative to T.

The reason why we believe that, e.g., Peano arithmetics and Zermelo–Fraenkel (ZF) set theory are consistent is that they have been used extensively and no inconsistency has been found in them.

Example

ZFC is consistent relative to ZF. In fact, also ZF¬C is consistent relative to ZF and hence the axiom of choice is independent of ZF.

Types in programs

Static type systems can be seen as proof systems, type checking as a proof checking, and type inference as a proof search.

> *Well-typed program cannot go wrong.*
>
> Robin Milner

There are various kinds of types

- the good—static types that guarantee no runtime errors of certain kind (Java, Haskell),
- the bad—static types that mainly decorate but do not guarantee no runtime errors (C, C++), or
- the ugly—dynamic types that detect errors at runtime (Python).

Rice's theorem

All non-trivial semantic properties of programs are undecidable.

Hence a fully automatic analysis of semantic properties, e.g. termination, is impossible.

Very brief summary (reasoning part)

We discussed various logical reasoning approaches. Namely

- propositional logic,
- (ground) first-order logic with theories (SMT), and
- first-order logic.

Although they differ significantly in their expressive power, they are connected and similar approaches are used in them. This also applies to proof assistants, which are usually based on even more expressive systems.

A question is how to use logic for programming? Moreover, in a reasonably efficient way. . . You will see!

Remark

All these methods are mostly symbolic, but the interplay between symbolic and statistical reasoning is a hot (and important) topic. However, it is still unclear how exactly this "symbiosis" will look like in the future. . .

Bibliography I

Avigad, Jeremy (2018). "The Mechanization of Mathematics". slides. URL: http://www.andrew.cmu.edu/user/avigad/Talks/mechanization_talk.pdf.

Harrison, John (2015). "Formalization of Mathematics for Fun and Profit.". slides. URL: https://www.cl.cam.ac.uk/~jrh13/slides/wollic-23jul15/slides.pdf.

Harrison, John, Josef Urban, and Freek Wiedijk (2014). "History of Interactive Theorem Proving". In: *Computational Logic*. Ed. by Jörg H. Siekmann. Vol. 9. Handbook of the History of Logic. North-Holland, pp. 135–214. DOI: https://doi.org/10.1016/B978-0-444-51624-4.50004-6.

Newcombe, Chris et al. (Mar. 2015). "How Amazon Web Services Uses Formal Methods". In: *Communications of the ACM* 58.4, pp. 66–73. ISSN: 0001-0782. DOI: 10.1145/2699417.

Nipkow, Tobias and Gerwin Klein (2014). *Concrete Semantics — with Isabelle/HOL*. Springer. ISBN: 978-3-319-10541-3. URL: http://concrete-semantics.org/.

Urban, Josef (2015). "Computer-Understandable Mathematics: Is It Coming?" slides. URL: http://www.karlin.mff.cuni.cz/~ssaos/2015/slides_urban.pdf.