Interactive Theorem Proving: What it is and What it can do

An Academic Perspective

Tobias Nipkow

Fakultät für Informatik
TU München

What is a Proof?

A traditional mathematical proof is a natural language *proof sketch*, possibly with many gaps:

Example

"... It is now easy to see that $p_1 * \cdots * p_n + 1$ is also a prime number."

A *complete formal proof* is a (possibly looong) list of atomic proof steps in some precisely fixed logic.

Example

A proof step: If P and $P \to Q$ then Q.

What is an ITP?

A system for the interactive construction
of a complete formal proof

The ideal: user sketches proof,
ITP checks proof and fills in details

The reality: user has to give many details

Why? Finding proofs cannot be automated completely

Checking complete formal proofs is automatic,
but complete formal proofs are very large

ITP tries to fill in details with
internal and external automatic provers

Interactive versus Automatic

Interactive: You can do *anything*,
but it takes work by an expert

Automatic: You can do only so much

USP: **ITPs have very expressive logics**

Reliability of ITPs

Most ITPs are based on a small(ish) kernel implementing some well-studied logic

Every proof in the ITP has to go through that kernel

USP: **ITP proofs are extremely trustworthy**

The proof assistant universe

HOL

Theorema

Agda

Lean

ACL2

Coq

Isabelle

Mizar

PVS

HOL Light

What do ITP proofs look like?

Isabelle:

theorem $\mathit{prime}(p) \Longrightarrow \sqrt{p} \notin \mathbb{Q}$
proof
  from *primep* have p: $1 < p$ by (simp add: prime_nat_iff)
  assume $\sqrt{p} \in \mathbb{Q}$
  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: $m^2 = p * n^2$
  proof
    from n and sqrt_rat have $m = |\sqrt{p}| * n$ by simp
    then have $m^2 = (\sqrt{p})^2 * n^2$
      by (auto simp add: power2_eq_square)
    also have $(\sqrt{p})^2 = p$ by simp
    also have $\cdots * n^2 = p * n^2$ by simp
    finally show ?thesis using of_nat_eq_iff by blast

$\vdots$

How to construct an ITP proof?

See presentation of Jaap Boender

Some landmark ITP proofs

Some landmark ITP proofs: mathematics

- Four Colour Theorem (Gonthier / Coq)
- Feit-Thompson Theorem (Gonthier *et al.* / Coq)
- Kepler Conjecture
  (Hales *et al.* / HOL Light + Isabelle)

Kepler Conjecture (1611)

Theorem (Hales 1998). No packing of 3-dimensional balls of the same radius has density greater than the *face-centered cubic packing*.

What does all of this have to do with
software verification?

Same tools apply . . .

Compiler Correctness

$compile :: source\_lang \rightarrow target\_lang$

Correctness theorem:

$semantics_{target}(compile(p)) = semantics_{source}(p)$

Nontrivial proof!

Needs complete formal language definitions!

C/Java/... program correctness

We can *prove* that a program satisfies
some pre-condition/post-condition specification.

Example correctness theorem:

$$\{\ \texttt{n < length(a) \&\& n < length(b)}\ \}$$
$$\texttt{copy(a,b,n)}$$
$$\{\ \forall\ \texttt{i < n. a[i] = b[i]}\ \}$$

Requires nontrivial infrastructure for reasoning about
Hoare-triples $\{P\}p\{Q\}$ is some language like C or Java

Some landmark ITP proofs: computer science

- C compiler (Leroy / Coq)
- Mini-Java compiler (Klein & Nipkow / Isabelle)
- OS micro kernel (Klein, Heiser *et al.* / Isabelle)

Effort of landmark ITP proofs

- 4 – 20 person years (???)
- 60.000 – 300.000 lines of proof

Much shorter a second time around!

Academic impact

ITPs are standard tools
in the programming language community

They are increasingly used in OS, Networks, Crypto, ...

From academia to industry:

AWS has a growing team of (a few years ago)
30+ *world class experts* using ITPs (and other tools)
to verify AWS software, eg encryption.