Theorem Provers and Inefficient Computation

Thomas Sewell

Dec 2024

Theorem Provers & Inefficient Computation

This lecture series is going to be about software:

- (interactive) theorem provers
- mechanized logics
- proof assistants

We'll start with an overview (what/why) and move on to more pragmatic concerns (how/how-not).

In the LSS style, this will mostly be lectures. There will be demonstrations. You are encouraged to try things out yourself if/when you have a computer available.

This is also a "light" overview of the field rather than a "deep" dive into current research.

Where Week 1 Finished Up

In the final hours of Week 1, we saw that an intuitionistic logic is equivalent to a programming language, and listed some implementations.

- Coq/Rocq
- Lean
- Agda
- Idris

Where Week 1 Finished Up

In the final hours of Week 1, we saw that an intuitionistic logic is equivalent to a programming language, and listed some implementations.

- HOL90
- HOL4
- HOL-Light
- HOL-Omega
- Isabelle

- Coq/Rocq
- Lean
- Agda
- Idris

- ACL2
- SAT/SMT solvers (Z3, CVC3, CVC4 etc)
- Vampire, E, SPASS & Metis

Quite a menagerie! Why?

First, let's look at a couple of them.

Isabelle/HOL

Isabelle is allegedly a generic intuitionistic prover.

- `https://isabelle.in.tum.de/`
- Isabelle/HOL is its most popular instance
- Isabelle/ZF and Isabelle/FOL are also used

Isabelle/HOL is a full featured classical theorem prover. It has substantial libraries and a lot of add-on features.

Isabelle has a built-in editor environment, and is a little easier to get started with than some competitors.

Isabelle/HOL

Isabelle is allegedly a generic intuitionistic prover.

- `https://isabelle.in.tum.de/`
- Isabelle/HOL is its most popular instance
- Isabelle/ZF and Isabelle/FOL are also used

Isabelle/HOL is a full featured classical theorem prover. It has substantial libraries and a lot of add-on features.

Isabelle has a built-in editor environment, and is a little easier to get started with than some competitors.

If you brought a computer, consider trying it out! (Needs some disk space and CPU time to set up.)

Holbert

Holbert is a simple theorem prover whose graphical interface is designed to resemble whiteboard mathematics.

- `https://liamoc.net/holbert/`
- Lead developer is Liam O'Connor (ANU)

This is a very different kind of tool to Isabelle/HOL or Lean.

Some Logic Puzzles

I took some quick notes of interesting logic puzzles from week 1:

- *Snoopy* is a *dog*. Every *dog* is *good*. Is *Snoopy* a *good dog*?

Some Logic Puzzles

I took some quick notes of interesting logic puzzles from week 1:

- *Snoopy* is a *dog*. Every *dog* is *good*. Is *Snoopy* a *good dog*?
- Formalise Heyting algebra and proof in propositional intuitionistic logic. Show every provable proposition is valid in every algebraic interpretation.

Some Logic Puzzles

I took some quick notes of interesting logic puzzles from week 1:

- *Snoopy* is a *dog*. Every *dog* is *good*. Is *Snoopy* a *good dog*?
- Formalise Heyting algebra and proof in propositional intuitionistic logic. Show every provable proposition is valid in every algebraic interpretation.
- $n > 0 \rightarrow (n + 1) > 0$
- Peirce's law: $((P \rightarrow Q) \rightarrow P) \rightarrow P$

Let's try it! Also, I may take requests.

Demo Time

During the lecture, this was an interactive demo of using Isabelle/HOL to prove Peirce's law.

Some of the LSS demo materials are at:

- `https://www.cse.unsw.edu.au/~tsewell/lss/`

There was also a quick attempt at the same fact using Holbert:

- `https://liamoc.net/holbert/`

What is a Theorem Prover?

We have seen some quite different ways to write a proof in a theorem prover:

- Write a proof term as a program (Lean).
- Write a collection of backward rule inferences (Isabelle/HOL).
- Appeal to an automatic method (Isabelle/HOL).
- Co-author a proof tree (Holbert).

What is a Theorem Prover?

We have seen some quite different ways to write a proof in a theorem prover:

- Write a proof term as a program (Lean).
- Write a collection of backward rule inferences (Isabelle/HOL).
- Appeal to an automatic method (Isabelle/HOL).
- Co-author a proof tree (Holbert).
- (bonus) State helper lemmas to guide an automated strategy (ACL2).

We have also seen some quite diverse logics. So, what is a proof tool?

What is a Theorem Prover?

What is a theorem prover (generically)? What is a logic?

- We've been studying this for (at least) a week now.

What is a Theorem Prover?

What is a theorem prover (generically)? What is a logic?

- We've been studying this for (at least) a week now.

A logic includes:

- A **syntax** of terms, e.g. *Snoopy*.
  - Some terms are **propositions**, e.g. *Mortal*(*Socrates*).
- A type of **sequents** or **proof states**, e.g. $\Gamma \vdash P$.
- A collection of **inference rules**, e.g. modus ponens.
- Some **axioms**.

There are plenty of variations, e.g. axioms could be inference rules, proof states can just be propositions, etc.

Mechanising a Logic

The mathematical presentation of each logic is **mechanical**.

- Despite predating the relevant **machines**.

We can mechanise, or implement, the logic in a computer:

- Encode the term **syntax**.
  - With standard design choices from programming-language theory.
- Implement the **inference rules**.
- Define the base **constants** and **axioms**.
- Track the **proof states** while checking the steps of a **proof**.

Mechanising a Logic

The mathematical presentation of each logic is **mechanical**.

- Despite predating the relevant **machines**.

We can mechanise, or implement, the logic in a computer:

- Encode the term **syntax**.
  - With standard design choices from programming-language theory.
- Implement the **inference rules**.
- Define the base **constants** and **axioms**.
- Track the **proof states** while checking the steps of a **proof**.
  - Now we can prove some theorems!

Some Design Choices

The shape of our logic implementation depends on our goals.

$$\text{Performance} \longleftrightarrow \text{Correctness}$$

$$\text{Generality} \longleftrightarrow \text{Simplicity}$$

$$\text{Classicism \& Pragmatism} \longleftrightarrow \text{Innovation \& Elegance}$$

$$\text{Education} \longleftrightarrow \text{various}$$

Thus the many tools in our menagerie.

Choice: Proofs as Programs

There are two ways we can use computer programs in the structure of our proofs.

Option 1: Curry-Howard isomorphism, proofs are programs.

- We saw this in detail in week 1.

Option 2: Automatic elaboration.

Why? For instance, how many steps does an inference take?

- Sometimes in week 1 we thought about the efficiency of deductions.

What if the user has a computer ally?

Choice: Proofs as Programs

There are two ways we can use computer programs in the structure of our proofs.

Option 1: Curry-Howard isomorphism, proofs are programs.

- We saw this in detail in week 1.

Option 2: Automatic elaboration.

Why? For instance, how many steps does an inference take?

- Sometimes in week 1 we thought about the efficiency of deductions.

What if the user has a computer ally? Thus the term **Proof Assistant**.

Elaboration and LCF

With an assistant, the user provides some minimal guidance and the tool unpacks the completed formal proof.

What is the absolute minimum? Are we familiar with Kolmogorov complexity?

In the **LCF** style, the proof script is a program and the prover/checker is just a library.

- Some **kernel** types are protected by the type system.
  - e.g. every **thm** is a theorem of the logic
- Design by Robin Milner in implementation of Edinburgh LCF.
- Led to the ML programming language family.

Tactics

One natural kind of assistant is a tactic:

```
tac :  (goal -> (goal list * validation))
```

A tactic examines the current goal (problem to be proven) and applies proof steps, yielding subgoals.

Why? Imagine we want to use the lemma $\forall x.\ x * 0 = 0$.

- This might be a built-in step.
- This might require many composition and transitivity steps.

Our Menagerie

LCF and HOL:

- HOL90
- HOL4
- HOL-Light
- HOL-Omega
- Isabelle

First-order/special-purpose:

- ACL2
- SAT/SMT solvers (Z3, CVC3, CVC4 etc)
- Vampire, E, SPASS & Metis

Type-Theory

i.e. proofs as programs

Tactics:

- Coq/Rocq
- Lean

Just Programs:

- Agda
- Idris

Recap

Recap. What is a mechanized logic?

- A language of terms & propositions.
- A language of proofs of propositions.
- A checker for these proofs.

There are lots of design decisions.

- What are our priorities?

Thanks for listening. Questions?

Bonus Puzzle

Just in case this runs far too quickly.

Bonus puzzle: Prove Peirce's law without tactics.