Seminar Lean

1. What is a Theorem Prover?

Def. (Theorem Prover):

A theorem prover is a software tool or system that verifies the correctness of a mathematical proof. As such, the theorem prover ensures to be reliable and accurate in the mathematical reasoning. The goal of a theorem prover is to validate a proof based on a set of axioms, rules of inference, and logical deduction.

2. Why do we use Theorem Prover?

A mathematical proof has two roles to fill: first, it convinces the reader that the statement is correct, and second, it explains why the proof is correct and the underlying ideas. A mathematical proof is constructed in small steps that can be verified easily. Mathematicians are in this regard sometimes lazy and skip proof steps because those are "definitely" true. But sometimes theorems are false, such that a "trivial" small step couldn't be verified. With that in mind, it can be even worse because when a wrong statement of a theorem is used in another proof, it is like pulling at the bottom card of a card house; it collapses. Getting back to the two roles of a proof, a theorem prover can not help us explain the proof or even explain the underlying ideas, but a theorem prover is definitely able to help us verify the correctness of a proof by trying to check mechanically that the theorem is true, and by doing so, we transform a previous informal proof into a formal proof. That is not an easy task because a proof in a textbook is not something a machine can understand, so a translation from an informal proof, which is written in a natural language, is a tedious job to do.

3. Small History of Theorem Provers

The earliest work on computer-assisted proof was in the 1950s. The first theorem provers were automated theorem provers (ATP), which are theorem provers that were supposed to prove an assertion fully automatically. Later, the idea came up for a more interactive arrangement where the user guided the proof. In order to guide a machine proof, there needs to be a language for the user to communicate the proof to the machine. Today, there are many theorem provers out there; some names you can recognize are Coq, HOL, Isabelle, etc. Some cornerstones for interactive theorem proving are Automath, LCF, Minzar and Martin-löf type theory.

4. What is Lean?

Lean is a functional programming language and an interactive theorem prover. Lean was developed at Microsoft Research in 2013. The idea behind Lean was to help mathematicians and engineers solve complex mathematical problems. Lean has a small, trusted kernel based on dependent type theory. Lean aims to bridge the gap between automated and interactive theorem proving by allowing automated tools in an interactive framework. It is a long-term effort and an open-source project with an active community, hosted on GitHub. The Lean community is actively contributing to Lean's mathematical library, Mathlib and has already digitized half of the undergrad curriculum. The two goals are to be done with the undergraded curriculum in less than 10 years and to prove Fermat's Last Theorem. Lean provides a platform for software verification and development, formalizing mathematics and education. Lean 4 is the latest version. Recently, a Lean team was able to help Field Medalist Peter Scholze confirm a theorem.

5. Applications

5.1 Lean in Mathematics

Bullet points.

Math

- Type theory
    - Lean is based on dependent type theory (Calculus of Constructions) (has a countable hierarchy of non-cumulative universes and inductive types.)

        - Dependent types can type more programs, they allow us to type things such as 3x10+1, while simply typed lambda calculus (STLC) can't. STLC only has one type constructor (\leftarrow), the function constructor, while dep. type theory has inductive types (and pi types - dependent functions) from which all other things follow.

            - Type constructors build new types from old ones.
            - Calc. o. Constr. is an extension of the Curry-Howard isomorphism in which terms in STLC are associated with deduction proofs in intutionistic (constructive) logic. C. o. C extends this to proofs with quantifiers. ("called propositions")
                - Intuitionistic logic is linked with computation.

        - Universes can be thought of as "big" types with elements which are also types.

            - Dep. type theory has a hierarchy of universes. Why? If, for example, you want to create a new type that references a universe, a new universe is implicitly defined.

        - The dep. type theory allows users to specify precise types for definitions and write formal proofs directly within the system.

        - The proofs written by the user can be verified with the internal type checker. Proofs are in itself, types. @

5.2 Lean in Computer Science

5.3 Lean in Teaching

6. Outro