Working Principles Of Proof Assistant

And Formalization Of Some Proofs In Agda

Ashwot Acharya, Bishesh Bohora, Supreme Chaudhary

Supervisor: Mr K.B Manandhar

Kathmandu University

Proof Assistants

What are proof assistant

proof assistant, are software more specifically a type of programming language thats allows us to formalize mathematical proofs in computer for digital verification.

Need of digital verification

- faster computation for complex problems
- many exceptional cases can be explored which would take mathematicians long time
  ex: The Kepler Conjecture's proof , which was so complex that verifying it manually would take 20 person-years, but proof assistants made this verification feasible and fast.

Logical foundation

- Based on logic (natural deduction, intuitionistic logic), λ-calculus, and type theory.
- Curry–Howard Correspondence:

$$\text{Propositions} \leftrightarrow \text{Types}$$
$$\text{Proofs} \leftrightarrow \text{Programs}$$

- Dependently Typed Languages: Types can depend on values, enabling encoding of properties and proofs.

Natural Deduction

- **Natural Deduction** is a rule-based system for deriving conclusions from assumptions in logic.
- Instead of using exhaustive truth tables, proofs are built step-by-step using inference rules.
- Example: Proving from $A \wedge (A \to \bot)$ that $\bot$ (contradiction) can be derived.
- Basis for how proof assistants check the logical structure of proofs.

Intuitionistic Logic

- **Intuitionistic Logic** formalizes constructive mathematics, where a statement is only true if a proof can be constructed.
- Omits some classical logic principles, such as the Law of Excluded Middle.
- Stronger requirement: to prove existence, a method or algorithm must be given.
- Proof assistants leverage this constructive approach for digital verification.

λ-Calculus and Type Theory

- λ-**Calculus**: A foundational system for defining and applying functions using abstraction and application.
- **Type Theory**: Assigns types to every term; ensures correctness of operations.
- **Curry–Howard Correspondence**:

$$\text{Propositions} \leftrightarrow \text{Types}$$
$$\text{Proofs} \leftrightarrow \text{Programs}$$

- *Dependent types* allow types to depend on values, expressing complex logical properties.

Architecture of proof assistant

Architecture of a Proof Assistant

- **Kernel**: Minimal, trustworthy codebase enforcing logical rules and validating proofs.
- **Tactic Engine**: Helps build and automate proofs step by step.
- **Formal Proof Language**: Rigorously expresses definitions, statements, and proofs.
- **Libraries**: Collections of verified mathematical foundations for reuse.
- **User Interface**: IDEs and plugins for interactive, efficient proof development.

Comparative Study

Agda

- **Dependently typed functional programming language and proof assistant.**
- No separate tactic layer—proofs are constructed explicitly and directly.
- Uses **bidirectional type checking**, minimal kernel, and integrates normalization.
- Emphasizes explicitness and formal clarity over automation.
- Best suited for foundational work and educational purposes.

Rocq (Coq)

- **Tactic-based interactive theorem prover** famous for large formalizations (e.g., Four Color Theorem).
- Kernel is based on Calculus of Inductive Constructions (CIC), written in Coq and extracted to OCaml.
- Proofs often built with backward reasoning using tactics—automation is heavily supported.
- Strong type-checking with bidirectional algorithms and conversion checking.
- Widely used for complex, real-world formal verification tasks.

Lean

- Developed at Microsoft, supports both **tactic-based and term-based** proofs.
- Kernel based on the Calculus of Inductive Constructions, implemented in C++/C.
- Allows seamless switching between tactic and term styles—user-friendly and flexible.
- Smart elaborator with coercion, backtracking, and overloading.
- Popular in research and education, especially for combinatorial and mathematical formalizations.

Formalization Of Some proofs

Thank you!