Introduction to Rocq — Arthur Azevedo de Amorim

Lecture 1 - *June 22, 2026*

1 General Information

Repository: `https://github.com/arthuraa/rocq-introduction`

Requires Rocq 9.0 and stdpp, see Rocq Platform docs. The lecture is delivered using the SSReflect proof language.

There will be a lab session tomorrow. This first lecture covers `basics.v`.

2 Definitions, Simplification, and Done

Common Rocq definitions include `Definition` (non-recursive functions), `Inductive` (algebraic data types or propositions), and `Fixpoint` (recursive functions). Fixpoints in Rocq cannot be arbitrarily recursing and must have one structurally decreasing argument.

Rocq has commands: `Compute` is one of them, used to evaluate an expression.

Proof statements usually follow `Lemma` or `Theorem`. Proof scripts then follow `Proof` and should be closed by `Qed`. `Admitted` marks unfinished proofs.

For `forall n, ...` in the goal, we use `intro n` to introduce the variable. This also works for implications: `intro H` on `A -> B` moves `A` as a hypothesis `H: A`. To introduce multiple entities, use `intros`.

Some goals can be proved by simplification. Use `rewrite /=` (or traditionally `simpl`). Trivial goals like reflexivity can be closed by `done`, which also conducts simplification. Thus `rewrite /=` can sometimes be omitted.

Compiled By:
Bas Laarakker, Songlin Jia, Ulises Torrella

3 Case Analysis, Tacticals

Fixpoint applications can only be simplified if the shape of the decreasing argument is known, *i.e.,* `add n (S m)` cannot be reduced. In such a case, we need case analysis `destruct n`. The proof obligation will then be split into two, where `add` calls can be simplified respectively.

The tactic `destruct` can be followed by an `as` clause to specify the names for destruction results. "Every time you do something like this, you write destruct n with a period, a baby kitten dies somewhere."

Some *tacticals* can help navigate with multiple subgoals. The dash `-` (actually also `+`, `*`, and their repetitions) focuses on a subgoal and hides the rest temporarily. The semicolon `tac1; tac2` allows you to apply `tac2` to all of the subgoals produced by `tac1`.

The tactic `induction` allows you to do inductive reasoning on recursive structures. It works similarly to `destruct`, but generates additional *induction hypotheses* for inductive cases. Also similarly, `induction` takes `as` clauses.

4 Logics

There is a cheatsheet `logic.v` on how to deal with logical connectives.

Broadly, for connectives ($\wedge$, $\vee$, $\exists$, $\iff$) occurring in hypotheses, use `destruct`. When $\exists$ occurs in the goal, use `exists`; $\vee$, `left` or `right`; $\wedge$ and $\iff$, use `split`.

5 Polymorphism

Definitions can take types as parameters, allowing polymorphism. Parameters defined in parentheses, *e.g.,* `(T: Type)`, require explicit instantiation, *i.e.,* must be provided at call sites. Those defined by curly brackets, *e.g.,* {`T`}, can be omitted.

The `Arguments` command can be used to inform Rocq to infer additional parameters. On the other hand, even for `nil` {`T`} which doesn't require providing `T`, one can use the `@`-notation for explicit instantiation, *e.g.,* `@nil nat`.