COMP2313
Formal Specification and Verification
Lecture 4: Rocq - Basics (continue) - Proof by Case Analysis

Asieh Salehi Fathabadi

University of Southampton

Prelude

MyEngagement code

437660

University of Southampton

Recall: Proof Techniques So Far

We have learned several proof techniques:

- **Simplification** with simpl
- **Reflexivity** with reflexivity
- **Rewriting** with rewrite
- **Introduction** with intros

Today we learn a new powerful technique: **Case Analysis**

- Used when a proof requires considering different cases
- Essential for reasoning about finite data types
- Implemented using the `destruct` tactic

The destruct Tactic

The `destruct` Tactic

```
Theorem plus_1_neq_0 : forall n : nat,
  (n + 1) =? 0 = false.
Proof.
  intros n.
  destruct n as [| n'] eqn:E.
  - (* n = 0 *)
    reflexivity.
  - (* n = S n' *)
    reflexivity.
Qed.
```

- `=?` checks equality and returns a `bool (true/false)`
- `destruct n` splits into cases: `0` and `S n'`
- `as [| n']` names the variables in each constructor case: empty before | means 0, n' after `|` names the inner value inside `S`
- `eqn:E` saves the case equation into the context as `E`
- `-` separate subgoals

Understanding Case Analysis

After `destruct n`:

Case 1: $\mathbf{n = O}$

```
n : nat
E : n = 0
==============================
(0 + 1) =? 0 = false
```

Case 2: $\mathbf{n = S\ n'}$

```
n, n' : nat
E : n = S n'
==============================
(S n' + 1) =? 0 = false
```

Both simplify to `false = false`.

Why Case Analysis Works

The `destruct` tactic leverages the structure of inductive types:

```
Inductive nat : Type :=
  | O          (* Zero *)
  | S (n:nat). (* Successor *)
```

- Every natural number is **either** `O` **or** `S n'` for some `n'`
- To prove something for all natural numbers, prove it for both cases
- No other cases are possible!

University of Southampton

Safety Analyis by Case Analysis

Big Idea: Exhaustive Case Analysis for Safety Verification

Case analysis splits a value into all possible constructors and solves each case separately. By handling every constructor exhaustively, we guarantee safety properties hold universally, if it's safe in all cases, it's always safe.

Key Points:

- **Follows the structure of the type:** bool gives 2 cases, nat gives 2 cases, day gives 7 cases, one case per constructor.
- **Peels off constructor layers:** destruct n turns an unknown n into concrete cases 0 and S n', making computation possible.
- **Exhaustive by design:** Rocq forces you to handle every constructor, you cannot miss a case, guaranteeing complete safety coverage.
- **Proves safety through impossibility** showing a bad case cannot occur in any constructor proves it never occurs.

Case Analysis on Booleans

Case Analysis on Booleans

```
Theorem negb_involutive : forall b : bool,
  negb (negb b) = b.
Proof.
  intros b.
  destruct b eqn:E.
  - (* b = true *)
    reflexivity.
  - (* b = false *)
    reflexivity.
Qed.
```

- Boolean has two cases: `true` and `false`
- No `as [| n']` needed because bool's constructors true and false have no inner values to name,
  unlike `S (n : nat)` which carries a value inside

Understanding Boolean Case Analysis

After `destruct b`:

Case 1: b = true

```
E : b = true
============================
negb (negb true) = true
```

Simplifies to: `true = true`

Case 2: b = false

```
E : b = false
============================
negb (negb false) = false
```

Simplifies to: `false = false`

Task: Prove the following theorem about boolean AND operation using only case analysis)

```
Theorem andb3_exchange :
  forall b c d, andb (andb b c) d
  = andb (andb b d) c.
Proof.
(* Replace Admitted with your Solution,
do not forget to add Qed. at the end*)
Admitted.
```

Join at:
vevox.app

ID:
143-242-789

Solution: andb3_exchange

```
Theorem andb3_exchange :
  forall b c d, andb (andb b c) d = andb (andb b d) c.
Proof.
  intros b c d. destruct b eqn:Eb.
  - destruct c eqn:Ec.
    { destruct d eqn:Ed.
      - reflexivity.
      - reflexivity. }
    { destruct d eqn:Ed.
      - reflexivity.
      - reflexivity. }
  - destruct c eqn:Ec.
    { destruct d eqn:Ed.
      - reflexivity.
      - reflexivity. }
    { destruct d eqn:Ed.
      - reflexivity.
      - reflexivity. }
Qed.
```

University of Southampton

Multiple Case Analysis

Case Analysis with Multiple Variables

```
Theorem andb_commutative : forall b c,
  andb b c = andb c b.
Proof.
  intros b c.   destruct b eqn:Eb.
  - (* b = true *)
    destruct c eqn:Ec.
     (* c = true *)
    +  reflexivity.
     (* c = false *)
    +  reflexivity.
  - (* b = false *)
    destruct c eqn:Ec.
     (* c = true *)
    +  reflexivity.
     (* c = false *)
    + reflexivity.
Qed.
```

Understanding Multiple Cases

With two boolean variables, we have 4 cases:

1. `b = true, c = true`: `andb true true = andb true true`
2. `b = true, c = false`: `andb true false = andb false true`
3. `b = false, c = true`: `andb false true = andb true false`
4. `b = false, c = false`: `andb false false = andb false false`

Each case simplifies and is proved by `reflexivity`.

Organizing Proofs

Bullet Styles

Rocq supports multiple bullet styles for organizing subgoals:

- - (single dash)
- + (plus sign)
- * (asterisk)
- { and } (braces)

```
Proof.
  intros b c.
  destruct b.
  { destruct c.
    { reflexivity. }
    { reflexivity. } }
  { destruct c.
    { reflexivity. }
    { reflexivity. } }
Qed.
```

Nested Bullets

Use different bullet levels for nested cases:

```
Proof.
  intros.
  destruct x.
  - (* x case 1 *)
    destruct y.
    + (* y case 1 *)
      reflexivity.
    + (* y case 2 *)
      reflexivity.
  - (* x case 2 *)
    destruct y.
    + (* y case 1 *)
      reflexivity.
    + (* y case 2 *)
      reflexivity.
Qed.
```

Comments in Proofs

Always add comments to clarify what each case represents:

```
Proof.
  intros n.
  destruct n as [| n'] eqn:E.
  - (* n = 0 *)
    (* Proof for base case *)
    reflexivity.
  - (* n = S n' *)
    (* Proof for successor case *)
    reflexivity.
Qed.
```

- Makes proofs easier to understand
- Helps when reviewing or debugging proofs
- Good practice for collaborative work

When Case Analysis is Not Enough

Limitations of Case Analysis

Consider trying to prove:

```
Theorem plus_n_O : forall n:nat,
  n = n + 0.
Proof.
  intros n.
  simpl.
  (* Nothing happens! *)
```

- `simpl` cannot reduce `n + 0` because `n` is a variable
- Case analysis with `destruct` only helps with the base case
- The successor case cannot be proved without the result for smaller numbers

Why Destruct Fails Here

Let's try using `destruct`:

```
Proof.
  intros n.
  destruct n as [| n'] eqn:E.
  - (* n = 0 *)
    simpl. reflexivity. (* This works! *)
  - (* n = S n' *)
    simpl.
    (* Goal: S n' = S (n' + 0) *)
    (* We're stuck! *)
```

- Base case: `0 = 0 + 0` is provable
- Successor case: `S n' = S (n' + 0)` requires knowing `n' = n' + 0`
- We need a more powerful technique: **induction**

Summary

Summary: Case Analysis

1. **The destruct Tactic**:
   - Splits proof into cases based on type constructors
   - Works on any inductive type
   - Essential for finite case analysis
2. **Organizing Proofs**:
   - Use bullets (-, +, *) to structure cases
   - Add comments for clarity
3. **When to Use**:
   - Finite number of cases
   - Each case provable independently
   - No dependency on previous cases

Key Takeaways

- **Case analysis is exhaustive**: Must cover all constructors
- **Cases are independent**: Each proved separately
- **Good for finite types**: Booleans, small enumerations
- **Structure matters**: Use bullets and comments
- **Not always sufficient**: Some proofs need induction

When case analysis fails: If proving case n requires knowing the result for case $n - 1$, you need induction (next lecture).

University of Southampton

Proof Strategies So Far

Technique	When to Use
`reflexivity`	When both sides are identical
`simpl`	To evaluate/reduce expressions
`rewrite`	To use equations/hypotheses
`destruct`	For case analysis on finite types

Next lecture: `induction` for recursive/infinite types

Additional Resources

- Software Foundations full textbook: `softwarefoundations.cis.upenn.edu`
- Rocq: `https://rocq-prover.org/`

Next Lecture

Topics for next time:

- Mathematical induction
- The `induction` tactic
- Proving properties for all natural numbers
- Helper lemmas and complex proofs

Questions?