Proving properties of programs

Yves Bertot

January 2015
Objectives

- Usual approach to removing bugs in programs: testing
- Write testing context, construct sample inputs, run
- This course: perform test with symbolic values
- Use quantification to introduce symbolic values
Examples of programs

Require Import Arith List.

Fixpoint evenb (n : nat) : bool :=
  match n with
  | 0 => true | S p => negb (evenb p)
  end.

Fixpoint max_list (l : list nat) : nat :=
  match l with
  | nil => 0
  | a::tl => max a (max_list tl)
  end.

Definition swap_first_two (l:list nat) : list nat :=
  match l with
  | a::b::tl => b::a::tl
  | _ => l
  end.
Reasoning on case expressions

- When a match appears in the goal
- Use `case`, `case_eq`, `destruct` to look separately at the various cases of execution
- demo time!
Impossible cases

- Impossibility can be expressed in several ways:
  1. premise or hypothesis true = false, 0 = 1, or nil = a::tl
  2. premise or hypothesis A <> A or A <> B when A actually equals B
  3. premise or hypothesis False

- Impossibility 1: discriminate

- Impossibility 2: case H
Mathematicians prove properties of natural numbers by induction.

For any predicate \( P \) on natural numbers:

- If \( P \ 0 \) holds
  - If one can deduce \( P \ (1 + n) \) from \( P \ n \) for any \( n \)

Then the properties holds for every natural number.

Only two cases, but infinity of results!

Like proof by cases, but with an induction hypothesis.
Using induction to prove properties on evendb

Demo time!
Non confusion of data-type constructors

- Constructors of data-types are manipulated as functions
- These functions have specific properties
  - Different constructors always yield different values
  - Each constructor is injective
- These properties are consequences of `match ... with ... end` behavior
- In proofs two tactics are provided to use these characteristics
  - `discriminate` to prove `0 <> S p` and goals of the same shape
  - `injection` to prove `S p = S q -> p = q`
Guiding computation

- Sometimes we want to replace sub-expressions with others that are equal
- If the system should be able to recognize it, use change
- If the system can’t recognize it, but you are sure you can prove it, use replace
- If you don’t want to write the result, use unfold or simpl