SOFTWARE VERIFICATION
AND COMPUTER PROOF
(lesson 2)

Enrico Tassi – Inria Sophia-Antipolis
Lessons

1. Writing programs in Coq
2. Writing proofs in Coq
3. Proving your recursive programs correct
4. Proofs in arithmetic and more
5. More data types
Today: writing proofs in Coq

- Morning: proofs with simple connectives
- Afternoon: equality and quantifiers
Propositions

- Propositions do have a type too: Prop
- For this morning we assume P Q R are predicates
  Variables P Q R : Prop.
- We focus on the logical connectives
Connectives

- Conjunction of $P$ and $Q$ is written $P \land Q$
- Disjunction of $P$ and $Q$ is written $P \lor Q$
- Negation of $P$ is written $\sim P$
- $P$ implies $Q$ is written $P \rightarrow Q$
## Connectives and related commands

<table>
<thead>
<tr>
<th>connective</th>
<th>use</th>
<th>prove</th>
</tr>
</thead>
<tbody>
<tr>
<td>P (\land) Q</td>
<td>destruct</td>
<td>split</td>
</tr>
<tr>
<td>P (\lor) Q</td>
<td>destruct</td>
<td>left, right</td>
</tr>
<tr>
<td>P (\rightarrow) Q</td>
<td>apply</td>
<td>intros</td>
</tr>
<tr>
<td>(\neg) P</td>
<td>case</td>
<td></td>
</tr>
</tbody>
</table>
Proof commands

1. I'll present them with a demo

2. You will have a short reference for the exercises

3. You will find more examples on the course notes

4. The full documentation can be found in the user manual of Coq https://coq.inria.fr/distrib/current/refman/
Demo
Equality

• The most useful predicate is equality, E.g. \( x = 3 \)

• Equality means:
  “the two expressions compute the same value”.

• E.g.
  \[
  3 + 7 = 10 \quad \text{(* nothing to prove *)}
  \]
  \[
  x=y \rightarrow y=z \rightarrow x=z \quad \text{(* something to prove *)}
  \]
## Commands for equality

<table>
<thead>
<tr>
<th>predicate</th>
<th>use</th>
<th>prove</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = b$</td>
<td>rewrite $\rightarrow \ldots$</td>
<td>reflexivity</td>
</tr>
<tr>
<td></td>
<td>rewrite $\leftarrow \ldots$</td>
<td></td>
</tr>
</tbody>
</table>
Quantifiers

- Universal quantification is written `forall (n : T), ...`
- Existential quantification is written `exists (n : T), ...`
- E.g. `forall (n : nat), exists (m : nat), leb n (n + m) = true.`
More on quantifiers

- Quantifiers can talk about almost everything, not just numbers, lists, etc. E.g.
  \[ \forall (P : \text{Prop}) (Q : \text{Prop}), ((P \rightarrow Q) \land P) \rightarrow Q \]

- Moreover we can use some syntactic sugar. E.g.
  \[ \forall (P Q : \text{Prop}), (P \rightarrow Q) \land (P \rightarrow Q) \]
  \[ A \leftrightarrow B \text{ instead of } (A \rightarrow B) \land (B \rightarrow A) \]
# Commands for quantifiers

<table>
<thead>
<tr>
<th>quantifier</th>
<th>use</th>
<th>prove</th>
</tr>
</thead>
<tbody>
<tr>
<td>forall n, P</td>
<td>apply</td>
<td>intros</td>
</tr>
<tr>
<td></td>
<td>apply ... with (n := v)</td>
<td></td>
</tr>
<tr>
<td>exists n, P</td>
<td>destruct</td>
<td>exists</td>
</tr>
</tbody>
</table>
Searching

- To complete the exercises you must search the library for some theorems about arithmetic
- They are too “hard” to be proved by you (for today)
- The command `SearchAbout pattern` finds all theorems whose statement matches pattern. E.g.
  - `SearchAbout (_ * _)`.
  - `SearchAbout (_ + _ = _)`.
  - `SearchAbout (_ + _ * _)`.
Demo
That is all for today!