# Introduction to Coq
# Part 2: Automation tactics

Yves Bertot

September 2023

# Automated tactics

- Logical workhorses
- Polynomial equalities
- Linear arithmetic
- Interval reasoning

# Simple logic reasoning with databases of theorems

- The tactic `auto` applies theorems from a given database
- Applicability is tested through pattern-matching
- Only theorems where unknowns in premise also appear in the conclusion
- Users can create their own databases of theorems
- Hypotheses of the current goal are automatically included
- The depth of proof search is limited to 5 by default
  - writing `auto` $n$ uses $n$ instead of 5
  - Information packed in conjunction is not used (look at `firstorder`)
  - Some Hints can be fine-tuned to force a more advanced behavior

# Example with auto and the local context

```
Lemma use_hyp_in_auto :
  forall P : nat -> Prop,
  (forall x y, P x -> P y -> P (x + y)) ->
  forall a b c d, P a -> P b -> P c -> P d ->
  P (a + ((b + c)) + d).
Proof.
auto.
Qed.
```

# Adding theorems to a database

```
#[export]
Hint Resolve Nat.Even_Even_add : ev_base.
#[export]
Hint Resolve Nat.Even_mul_r : ev_base.
#[export]
Hint Resolve Nat.Even_mul_l : ev_base.

Lemma Even2 : Nat.Even 2.
Proof. exists 1. reflexivity. Qed.

#[export]
Hint Resolve Even2 : ev_base.
```

If you only want to improve the default database, its name is core

# Using a database

```
Lemma use_thms_in_auto n m :
  Nat.Even ((((2 * n + m * 2) + (2 * n + m * 2)) +
            ((2 * n + m * 2) + (2 * n + m * 2))) +
            (((2 * n + m * 2) + (2 * n + m * 2)) +
            ((2 * n + m * 2) + (2 * n + m * 2)))).
Proof.
auto 20 with ev_base.
Qed.
```

# Intuitionistic tautologies

- Some automatic tools expand all conjunctions and disjunctions in the context before calling `auto`
- intuition, tauto, firstorder

# Reasoning about numerical equalities

- Some automatic tactic solves all equality proofs about addition, multiplication, neutral elements, and subtraction (when possible)
- When it fails, three things may occur
  - The equality you want to prove is simply not true
  - You have more knowledge than what the `ring` tactic has access to
  - You are actually in the `nat` semi-ring, and your formula contains subtractions
- This tactic is also extensible: you can adapt it to your pet ring

# ring example

```
Require Import ZArith.
Open Scope Z_scope.

Lemma quartic_sub a b :
  (a - b) ^ 4 =
  a ^ 4 - 4 * a * b ^ 3 + 6 * a ^ 2 * b ^ 2 -
  4 * a ^ 3 * b + b ^ 4.
Proof.
ring.
Qed.
```

# An example failure in nat

```
Lemma quartic_sub_nat (a b : nat) :
  (a - b) ^ 4 =
  a ^ 4 - 4 * a * b ^ 3 + 6 * a ^ 2 * b ^ 2 - 4 * a ^ 3 * b
Proof.
Fail ring.
cbv [Nat.pow].
Fail ring.
```

Thus statement is simply not true

```
Abort.
```

# More structure: automatic equality proofs in fields

- There is also a `field` tactic
- It can also handle division, but the divisors have to be proved non-zero
- It tries to prove non-zero conditions directly
- May produce extra goals

# Example use of field

```
Require Import Reals Lra.

Open Scope R_scope.

Lemma sum_step n : n * (n + 1) / 2 + (n + 1) =
    (n + 1) * (n + 2) / 2.
Proof.
field.
Qed.

Lemma gen_sum_step n f : f <> 0 ->
  n * (n + 1) / f + (n + 1) =
  (n + 1) * (n + f) / f.
Proof.
intros fn0.
field.
auto.
Qed.
```

# Numeric comparison: linear arithmetic

- You have to request for them, but there are tactics to reason about comparisons between linear formulas
- The tactics are named `lia` (for integers) or `lra` for real numbers

# Example proof with `lra`

```
Lemma example_linear_arithmetic x y :
  (x * x) / 2 + 1 <= y ->
  y <= - 2 * (x * (x * 1)) + 1 ->
  y <= (x * x) + 10 ->
  (x * x) <= y.
Proof.
intros half_plane_1 half_plane_2 half_plane_3.
lra.
Qed.
```

# linear arithmetics only knows integer constants

- Mathematical constants like $e$ or $\pi$ are treated as variables
- For use with `lra`, the `interval` tactic can be used

# Example using interval

```
From Interval Require Import Tactic.

Lemma example_with_PI x : PI <= x -> x < 4 * x.
Proof.
intros xgepi.
Fail lra.
interval_intro PI.
lra.
Qed.
```

# Interval answers interval questions with rational bounds

In my experiments, it refuses questions with strict comparisons

```
Lemma approx_pi : 3.14 <= PI <= 3.15.
Proof.
interval.
Qed.
```