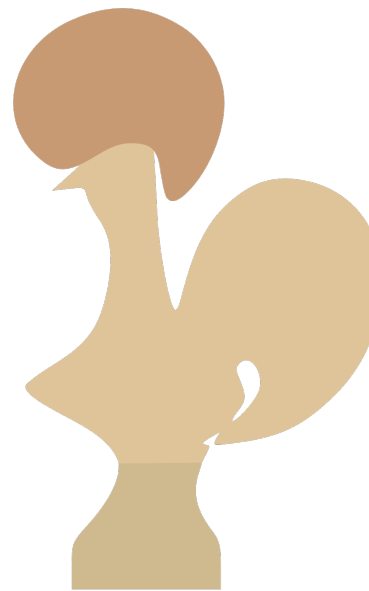


Coq Proof Assistant	LeanProver	same	requires mathlib
Theorem	theorem	exact	left, right
admit	sorry	apply	ring
reflexivity	rfl	intros	exists (use)
rewrite H	rw [H]	assumption	lia (linarith)
rewrite <- H	rw [<- H]	unfold	
simpl, cbn, auto	simp, dsimp	contradiction	
destruct, case, elim	cases	constructor	
discriminate	contradiction	induction	
remember, assert, pose	have	repeat	
subst	subst_vars	try	
;	<;>	refine	
-	\.	specialize	
A B	(first A B)	clear	
in	at	trivial	
generalize dependent	revert		
split	apply And.intro		
symmetry	apply Eq.symm		
f_equal	apply congrArg		



LEAN
THEOREM PROVER

```
theorem plus_assoc:
  forall x y z: Nat,
    (x+y)+z = x+(y+z) := by
  intros x y z
  induction x with
  | zero => simp
  | succ x IH =>
    repeat rw [Nat.succ_add]
    rw [IH]
```

```
theorem add_comm:
  ∀ a b: Prop,
    a /\ b -> b /\ a := by
  intros a b H
  cases H with
  | intro H1 H2 =>
    apply And.intro
    case left => exact H2
    case right => exact H1
```

```
theorem add_comm':
  ∀ a b: Prop,
    a /\ b -> b /\ a := by
  intros a b H
  have H1 := H.left
  have H2: b := H.right
  exact And.intro H2 H1
```

```
example (x y: Nat):
  succ x ≤ succ y
  <-> x ≤ y := by
  apply Iff.intro
  case mp =>
    apply ...
  case mpr =>
    apply Nat.succ_le_succ
```