<table>
<tr><th>Coq Proof Assistant</th><th>LeanProver</th></tr>
<tr><td>Theorem</td><td>theorem</td></tr>
<tr><td>admit</td><td>sorry</td></tr>
<tr><td>reflexivity</td><td>rfl</td></tr>
<tr><td>rewrite H</td><td>rw [H]</td></tr>
<tr><td>rewrite <- H</td><td>rw [<- H]</td></tr>
<tr><td>simpl, cbn, auto</td><td>simp, dsimp</td></tr>
<tr><td>destruct, case, elim</td><td>cases</td></tr>
<tr><td>discriminate</td><td>contradiction</td></tr>
<tr><td>remember, assert, pose</td><td>have</td></tr>
<tr><td>subst</td><td>subst_vars</td></tr>
<tr><td>;</td><td><;></td></tr>
<tr><td>-</td><td>\.</td></tr>
<tr><td>A|B</td><td>(first|A|B)</td></tr>
<tr><td>in</td><td>at</td></tr>
<tr><td>generalize dependent</td><td>revert</td></tr>
<tr><td>split</td><td>apply And.intro</td></tr>
<tr><td>symmetry</td><td>apply Eq.symm</td></tr>
<tr><td>f_equal</td><td>apply congrArg</td></tr>
</table>

same
exact
apply
intros
assumption
unfold
contradiction
constructor
induction
repeat
try
refine
specialize
clear
trivial

requires mathlib
left, right
ring
exists (use)
lia (linarith)

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
```

Authors: Paul Cadman, Gregor Feierabend, Walter Schulze