# Proving Termination of the Hydra Battle in Rocq

## A Journey into Interactive Theorem Proving and Termination Proofs

**John Alejandro González-González**[1]
**Andrés Sicard-Ramírez**[2]

[1] *Mathematical Engineering, Universidad EAFIT, jagonzale4@eafit.edu.co*

[2] *School of Applied Sciences and Engineering, Universidad EAFIT, asr@eafit.edu.co*

July, 2025

# Outline

# Lists in Rocq

Rocq provides a built-in list type, defined inductively:

```
Inductive list (A : Type) : Type :=
  | nil  : list A
  | cons : A -> list A -> list A.
```

We can then define functions over lists, for example, concatenation:

```
Fixpoint app {A : Type} (l1 l2 : list A) : list A :=
  match l1 with
  | nil      => l2
  | cons x t => cons x (app t l2)
  end.
```

Notation for convenience:

```
Notation "x :: l"   := (cons x l)      (at level 60, right associativity).
Notation "[]"       := nil.
Notation "l1 ++ l2" := (app l1 l2)    (at level 60, right associativity).
```

Now you can write expressions like: `1 :: 2 :: [] ++ [3; 4]`.

# Rose Trees and Height in Rocq

Define a rose (multi-way) tree type:

```
Inductive RoseTree : Type :=
  | Node : nat -> list RoseTree -> RoseTree.
```

Measure its *height* (max depth of nodes):

```
Fixpoint height (t : RoseTree) : nat :=
  match t with
  | Node _ children =>
      1 + fold_right max 0 (map height children)
  end.
```

# Rose Tree Example with Height

Consider this rose tree of height 3:

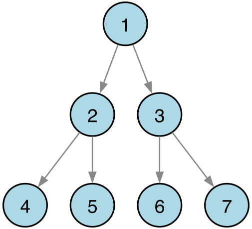



Figure: A binary tree with 7 nodes and height 3.

Its Coq representation as a rose tree:

```
Definition example_rose : RoseTree :=
  Node 1 [
    Node 2 [Node 4 []; Node 5 []];
    Node 3 [Node 6 []; Node 7 []]
  ].
Compute height example_rose. (* = 3 *)
```

# The Hercules Hydra Battle

The Hercules Hydra Battle is a famous history from Greek mythology, where Hercules faces the learnean Hydra, a serpent-like creature with multiple heads. Each time a head is cut off, two more grow back in its place.

Figure: Hercules and the Hydra of Lerna (1876). Oil on canvas, 179.3 x 154 cm. Art Institute of Chicago. Gustave Moreau (1826-1898).

# A Mathematical Version of the Hydra Battle

- A mathematical version of the Hydra myth. [1]
- Start with a rooted tree (the Hydra). Leaves are "heads".
- Choose a head, cut it off.
- If the cut head was height 1 from the root, the head is just removed.
- If the cut head was height $> 1$ from the root:
  - Let $G$ be its grandparent node.
  - Add $n$ new branches to $G$, each identical to the node removed.
- The game continues until there's only one head left (the root).

We want to prove that the game eventually terminates.

Given any hydra, the game eventually terminates.

# Sketch of the Termination Proof (Based on Rocq Formalization)

We prove termination of the Hydra game using a lexicographic measure over lists of natural numbers.

# Modeling the Hydra: Counts by Depth

We represent each Hydra tree by a *list of naturals*, where the *i*-th element is the number of nodes at depth *i*.

```
Fixpoint merge_counts (l1 l2 : list nat) : list nat :=
  match l1, l2 with
  | [], l => l
  | l, [] => l
  | x1 :: xs1, x2 :: xs2 => (x1 + x2) :: merge_counts xs1 xs2
  end.

Fixpoint _count_levels (t : RoseTree) : list nat :=
  match t with
  | Node _ children =>
      let child_counts := map _count_levels children in
      let merged := fold_left merge_counts child_counts [] in
      1 :: merged
  end.

Definition count_levels (t : RoseTree) : list nat := rev (_count_levels t).
```

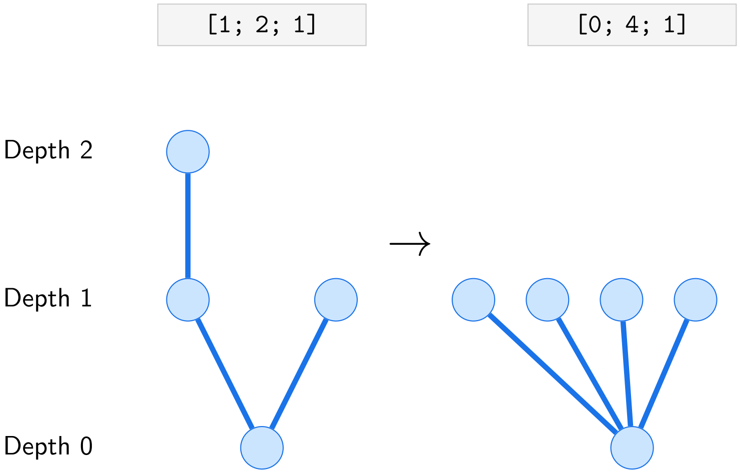
# Internal Cut: step_internal

An *internal* Hydra step cuts a head at depth $> 1$, redistributing copies to its grandparent:

```
Inductive step_internal (n : nat) : list nat -> list nat -> Prop :=
| StepInternalLong :
    forall prefix x y suffix,
      x > 0 ->
      length suffix >= 1 ->
      step_internal n (prefix ++ x :: y :: suffix)
                      (prefix ++ (x - 1) :: (y + n) :: suffix)
| StepInternalTwo :
    forall prefix x y,
      x > 0 ->
      step_internal n (prefix ++ x :: [y])
                      (prefix ++ (x - 1) :: [y]).
```

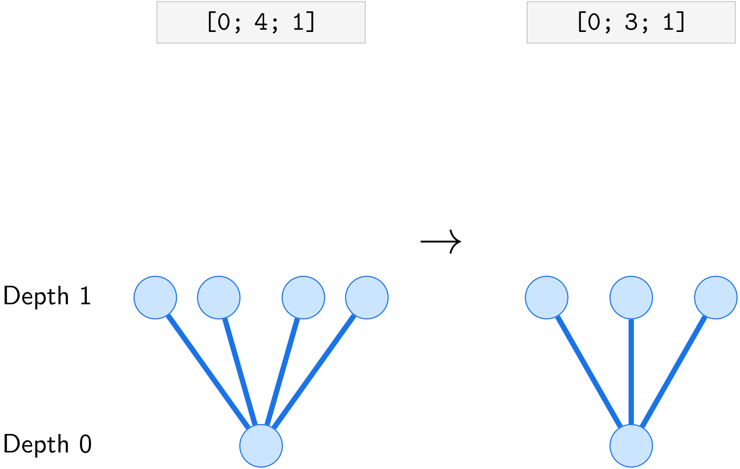
We decrease the count at the cut depth and increase at the parent's depth.

# Beheading at height $>$ 1 from the root




**test_step_long**

# Beheading at height = 1 from the root




**test_step_two**

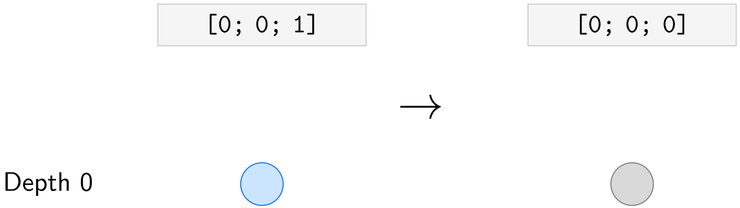
# Final Cut: step_final

A *final* step removes a last head when only one depth remains:

```
Inductive step_final : list nat -> list nat -> Prop :=
| StepFinal :
    forall prefix x,
      x > 0 ->
      step_final (prefix ++ [x]) (prefix ++ [x - 1]).
```

This simply subtracts one from the last nonzero entry.

# Beheading at the root




**step_final: the game ends!**

# Step Definition: step

We define a step as either an internal or final move:

```
Inductive step (n : nat) : list nat -> list nat -> Prop :=
| Step_internal_case :
    forall l1 l2,
      step_internal n l1 l2 ->
      step n l1 l2
| Step_final_case :
    forall l1 l2,
      step_final l1 l2 ->
      step n l1 l2.
```

This captures both types of moves in the Hydra game.

# Termination: step_done

An *empty* list indicates the Hydra is done:

```
Inductive step_done : list nat -> Prop :=
| StepDoneEmpty :
    step_done []
| StepDoneNonEmpty :
    forall l,
      l <> [] ->
      (forall x, In x l -> x = 0) ->
      step_done l.
```

This means all heads have been cut, and no more moves are possible.

# Complete Transition: step or done

We combine steps and termination tests:

```
Inductive step_or_done (n : nat) : list nat -> list nat -> Prop :=
| Step_case :
    forall l1 l2,
      step n l1 l2 ->
      step_or_done n l1 l2
| Done_case :
    forall l,
      step_done l ->
      step_or_done n l l.
```

Thus each configuration either evolves or is declared done.

# Length Preservation Lemmas

Both internal and final moves preserve the list length:

```
Lemma step_internal_preserves_length :
forall n l1 l2, step_internal n l1 l2 -> length l1 = length l2.

Lemma step_final_preserves_length :
forall l1 l2, step_final l1 l2 -> length l1 = length l2.

Theorem step_preserves_length :
forall n l1 l2, step n l1 l2 -> length l1 = length l2.
```

This invariant ensures our lexicographic comparison is well-defined.

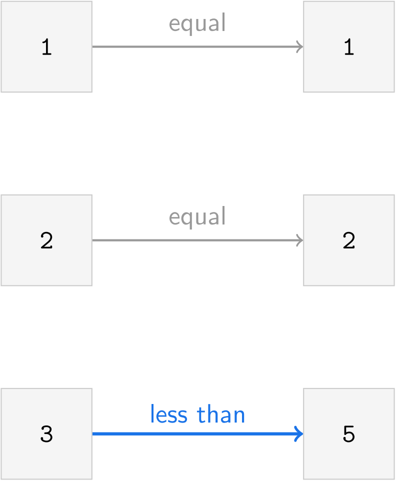
# Lexicographic Measure: lex_lt

Define a well-founded lex order over lists of naturals:

```
Inductive lex_lt : list nat -> list nat -> Prop :=
| lex_lt_nil : forall x xs, lex_lt [] (x :: xs)
| lex_lt_cons_lt : forall x y xs ys,
    x < y -> lex_lt (x :: xs) (y :: ys)
| lex_lt_cons_eq : forall x xs ys,
    lex_lt xs ys -> lex_lt (x :: xs) (x :: ys).
```

We compare first on heads, then recursively on tails.

# Lexicographic Comparison: [1; 2; 3] < [1; 2; 5]



# Each Step Decreases the Measure

Prove that every move strictly lowers the lex order:

```
Lemma step_internal_decreases_lex :
  forall n l1 l2,
    step_internal n l1 l2 ->
    lex_lt l2 l1.

Lemma step_final_decreases_lex :
  forall l1 l2,
    step_final l1 l2 ->
    lex_lt l2 l1.

Theorem step_decreases_measure :
  forall n l1 l2,
    step n l1 l2 ->
    lex_lt l2 l1.
```

This guarantees progress towards termination under a well-founded relation. But, is it well-founded?

# Proving Well-Foundedness of `lex_lt`

To complete the termination proof, we need to establish that `lex_lt` is well-founded:

```
Theorem lex_lt_wf : well_founded lex_lt.
```

**Status:** Pending work

Once proven, combined with our decrease lemma, this establishes that the `step` relation is well-founded, guaranteeing termination.

# Progress Property: Every Hydra Can Make a Step

We also need to prove that every non-terminal Hydra configuration can make progress:

```
Theorem hydra_progress :
  forall n l1,
    exists l2, step_or_done n l1 l2.
```

**Proof strategy:**

- Case analysis on the structure of the list `l1`
- If `l1` satisfies `step_done`, then `step_or_done n l1 l1`
- Otherwise, construct a valid step using `step_internal` or `step_final`

**Status:** Pending work

This property ensures that the game never gets "stuck" in a non-terminal state.

# Complete Termination Theorem

Combining all our results yields the main theorem:

### Hydra Battle Termination

For any natural number $n$ and any initial Hydra configuration $l_0$, the Hydra battle terminates in finitely many steps.

**Proof outline:**

1. `lex_lt` is well-founded (pending)
2. Every step decreases the lexicographic measure (proven)
3. Every configuration can make progress (pending)
4. Therefore, no infinite sequence of steps exists

This formalizes the classical result that Hercules always wins the (simplified) Hydra battle, regardless of the initial configuration or the value of $n$.

# Further Reading / Resources


- The Rocq (formerly Coq) development team, "Rocq Prover," https://rocq-prover.org/
- B. Pierce *et al.*, "Basics," in *Software Foundations*, https://softwarefoundations.cis.upenn.edu/lf-current/Basics.html
- A. Chlipala, "Universes," in *Certified Programming with Dependent Types*, http://adam.chlipala.net/cpdt/html/Universes.html
- YouTube: "The Hydra vs. Hercules – Numberphile," https://www.youtube.com/watch?v=prURA1i8Qj4
- P. Casteran, "Hydras&Co," https://rocq-community.org/hydra-battles/doc/hydras.pdf
- L. Kirby and J. Paris, "Accessible independence results for Peano arithmetic," *Bull. London Math. Soc.*, vol. 14, pp. 285–293, 1982.