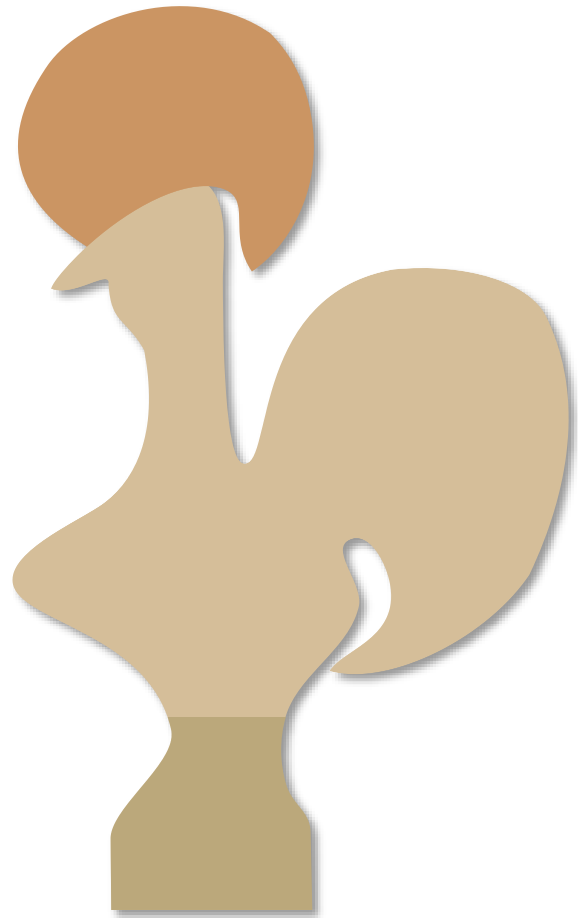
The HoTT/HoTT Library in Coq Designing for Speed

Jason Gross
Massachusetts Institute of Technology

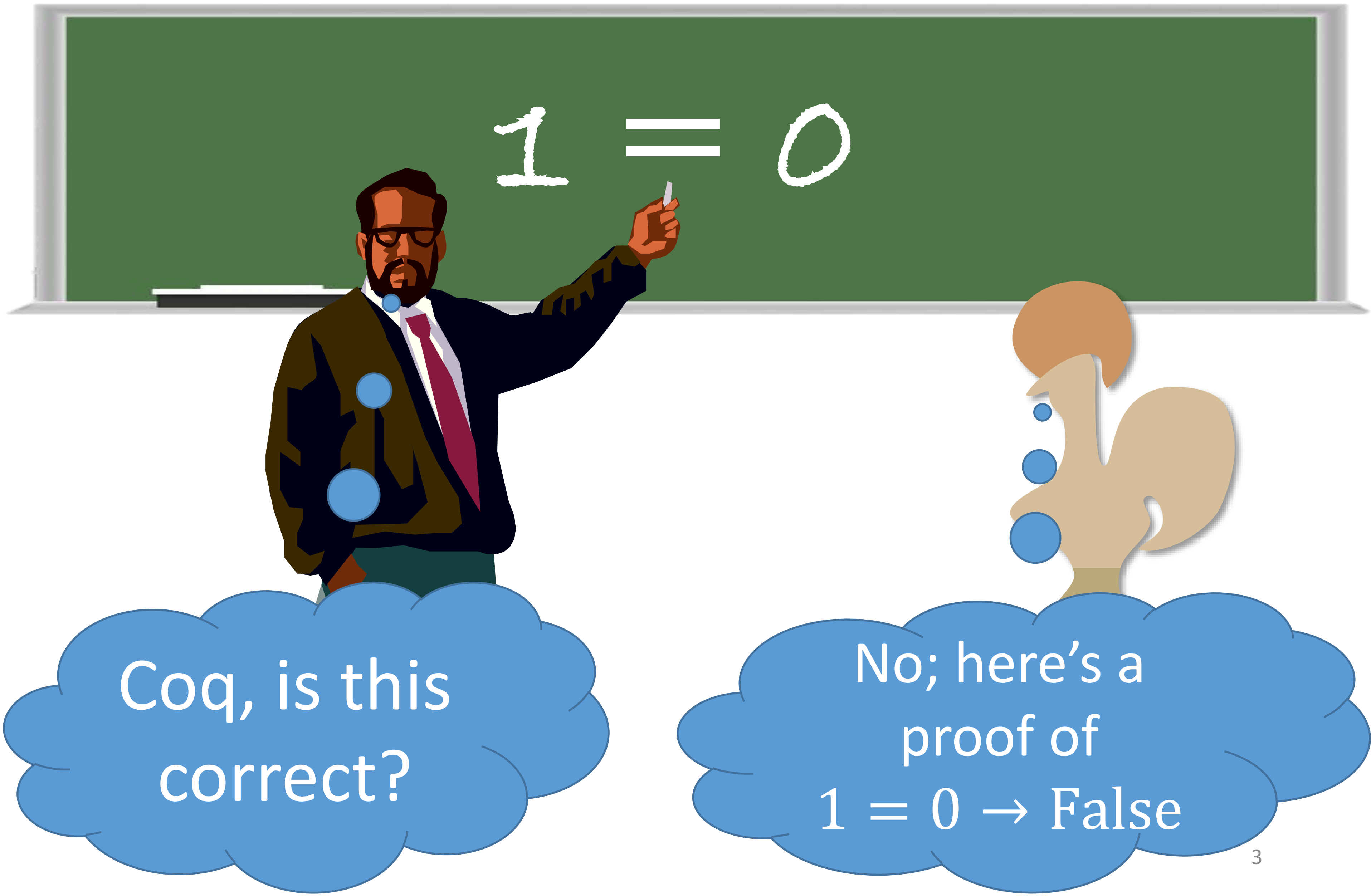
For ICMS 2016, adapted from ITP 2014 presentation

Category theory work done with Adam Chlipala and David I. Spivak
HoTT/HoTT library additionally co-authored by Andrej Bauer, Peter LeFanu Lumsdaine, Mike Shulman, Bas Spitters, and includes contributions from Assia Mahboubi, Marc Bezem, Kristina Sojakova, Daniel R. Grayson, Gaetan Gilbert, Matthieu Sozeau, Jérémy Ledent, Kevin Quirin, Steve Awodey, Cyril Cohen, Egbert, Benedikt Ahrens, Edward Z. Yang, Georgy Dunaev, Jesse C. McKeown, Simon Boulier, Alexander Karpich, Jelle Herold, John Dougherty, Matěj Grabovský, Michael Nahas, and Yves Bertot

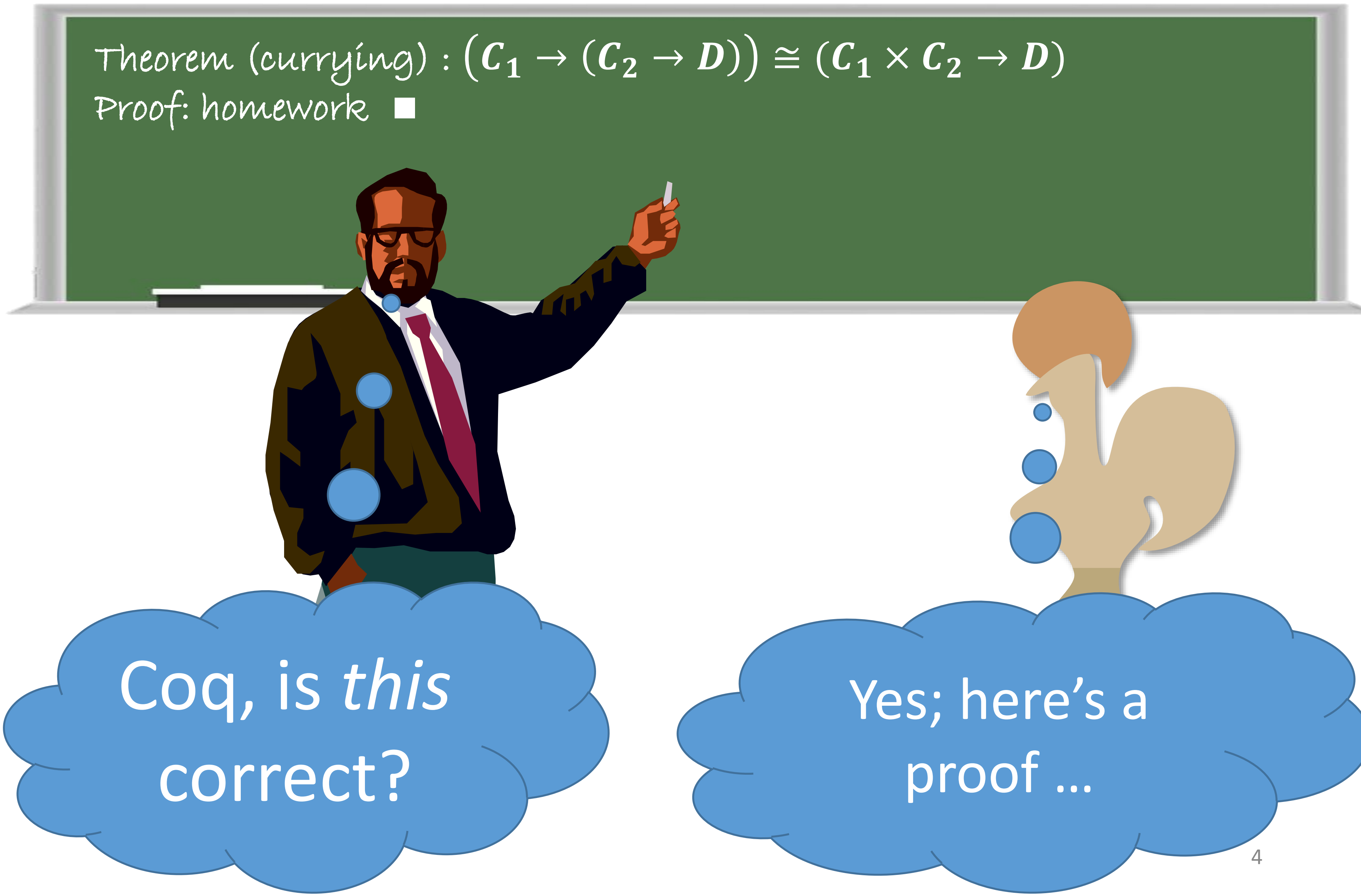
How should theorem provers work?



How theorem provers should work:



How theorem provers should work:



How theorem provers should work:

Theorem (currying) : $\left(\boldsymbol{C_1} \to (\boldsymbol{C_2} \to \boldsymbol{D})\right) \cong (\boldsymbol{C_1} \times \boldsymbol{C_2} \to \boldsymbol{D})$

Proof: homework ■

Theorem currying : $\left(C_1 \ \to (C_2 \to D)\right) \cong (C_1 \times C_2 \to D)$.

Proof.

  trivial.

Qed.

How theorem provers should work:

Theorem (currying) : $\left(C_1 \to (C_2 \to D)\right) \cong (C_1 \times C_2 \to D)$

Proof: $\to$: $F \mapsto \lambda\,(c_1, c_2).\ F(c_1)(c_2)$; morphisms similarly

$\leftarrow$: $F \mapsto \lambda\, c_1. \lambda\, c_2. F(c_1, c_2)$; morphisms similarly

Functoriality, naturality, and congruence: straightforward. ■

Theorem currying : $\left(C_1 \to (C_2 \to D)\right) \cong (C_1 \times C_2 \to D)$.

Proof.

esplit.

{ by refine (λ_F $(F \mapsto (\lambda_F\ (c \mapsto F_0\ c_1\ c_2)))$). }

{ by refine (λ_F $(F \mapsto (\lambda_F\ (c_1 \mapsto (\lambda_F\ (c_2 \mapsto F_0\ (c_1, c_2))))))$). }

all: trivial.

Qed.

How theorem provers should work:

Theorem (currying) : $\big(C_1 \to (C_2 \to D)\big) \cong (C_1 \times C_2 \to D)$

Proof: $\to$: $F \mapsto \lambda\,(c_1, c_2).\ F(c_1)(c_2)$; morphisms similarly

$\leftarrow$: $F \mapsto \lambda\,c_1.\lambda\,c_2.F(c_1, c_2)$; morphisms similarly

Functoriality, naturality, and congruence: straightforward. ∎

Theorem currying : $\big(C_1 \ \to (C_2 \to D)\big) \cong (C_1 \times C_2 \to D)$.

Proof.

esplit.

{ by refine (λ_{F} $(F \mapsto (\lambda_{\mathrm{F}}\ (c \mapsto F_{\mathrm{o}}\ c_1\ c_2)\ (s\ d\ m \mapsto (F_{\mathrm{o}}\ d_1)_{\mathrm{m}}\ m_2 \circ (F_{\mathrm{m}}\ m_1)_{\mathrm{o}}\ s_2))$

$(F\ G\ T \mapsto (\lambda_{\mathrm{T}}\ (c \mapsto T\ c_1\ c_2))))$. }

{ by refine (λ_{F} $(F \mapsto (\lambda_{\mathrm{F}}\ (c_1 \mapsto (\lambda_{\mathrm{F}}\ (c_2 \mapsto F_{\mathrm{o}}\ (c_1, c_2))\ (s\ d\ m \mapsto F_{\mathrm{m}}\ (1, m))))$

$(F\ G\ T \mapsto (\lambda_{\mathrm{T}}\ (c_1 \mapsto (\lambda_{\mathrm{T}}\ (c_2 \mapsto T\ (c_1, c_2))))))$. }

all: trivial.

Qed.

How theorem provers do work:

Theorem (currying) : $\big(C_1 \to (C_2 \to D)\big) \cong (C_1 \times C_2 \to D)$

Proof: $\to$: $F \mapsto \lambda\,(c_1, c_2).\ F(c_1)(c_2)$; morphisms similarly

$\leftarrow$: $F \mapsto \lambda\, c_1 . \lambda\, c_2 . F(c_1, c_2)$; morphisms similarly

≈ 0 s

Functoriality, naturality, and congruence: straightforward. ■

17 s

2m 46 s !!! (5 s, if we use UIP)

Theorem currying : $\big(C_1 \ \to (C_2 \to D)\big) \cong (C_1 \times C_2 \to D)$.

Proof.

esplit.

{ by refine (λ_{F} $(F \mapsto (\lambda_{\mathrm{F}}\ (c \mapsto F_{\mathrm{o}}\ c_1\ c_2)\ (s\ d\ m \mapsto (F_{\mathrm{o}}\ d_1)_{\mathrm{m}}\ m_2 \circ (F_{\mathrm{m}}\ m_1)_{\mathrm{o}}\ s_2))$

$(F\ G\ T \mapsto (\lambda_{\mathrm{T}}\ (c \mapsto T\ c_1\ c_2))))$. }

{ by refine (λ_{F} $(F \mapsto (\lambda_{\mathrm{F}}\ (c_1 \mapsto (\lambda_{\mathrm{F}}\ (c_2 \mapsto F_{\mathrm{o}}\ (c_1, c_2))\ (s\ d\ m \mapsto F_{\mathrm{m}}\ (1, m))))$

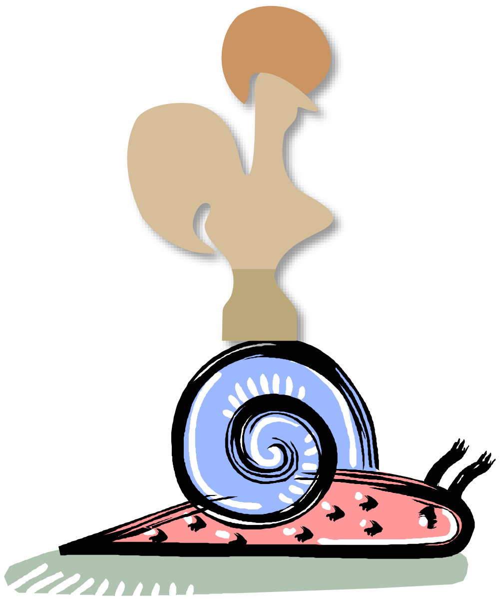
$(F\ G\ T \mapsto (\lambda_{\mathrm{T}}\ (c_1 \mapsto (\lambda_{\mathrm{T}}\ (c_2 \mapsto T\ (c_1, c_2))))))$. }

all: trivial.

Qed.

Performance is important!

If we're not careful, obvious or trivial things can be very, very slow.



Why you should listen to me

Theorem : You should listen to me.

Proof.

by experience.

Qed.

Why you should listen to me

Category theory in Coq: https://github.com/HoTT/HoTT (subdirectory theories/categories):

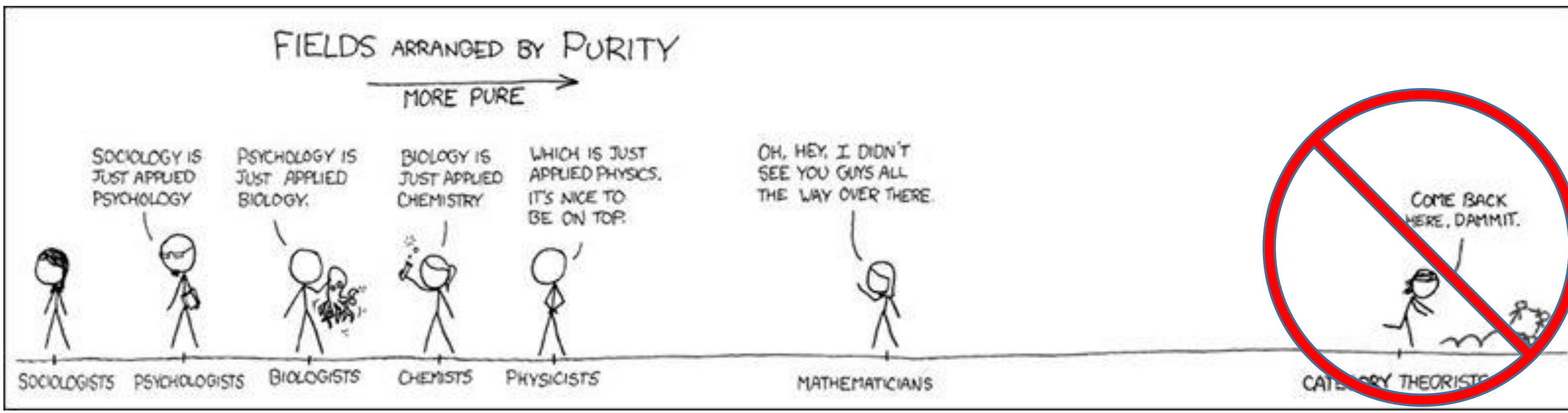
Concepts Formalized:

- 1-precategories (in the sense of the HoTT Book)
- univalent/saturated categories (or just categories, in the HoTT Book)
- functor precategories $C \to D$
- dual functor isomorphisms Cat → Cat; and $(C \to D)^{\mathrm{op}} \to (C^{\mathrm{op}} \to D^{\mathrm{op}})$
- the category Prop of (U-small) hProps
- the category Set of (U-small) hSets
- the category Cat of (U-small) strict (pre)categories (strict in the sense of the objects being hSets)
- pseudofunctors
- pseudonatrual transformations
- (op)lax comma categories
- profunctors
    - identity profunctor (the hom functor $C^{\mathrm{op}} \times C \to \mathrm{Set}$)
- adjoints
    - equivalences between a number of definitions:
        - unit-counit + zig-zag definition
        - unit + UMP definition
        - counit + UMP definition
        - universal morphism definition
        - hom-set definition
    - composition, identity, dual
    - pointwise adjunctions in the library, $G^E \dashv F^C$ and $E^F \dashv C^G$ from an adjunction $F \dashv G$ for functors $F: C \leftrightarrows D: G$ and E a precategory
- Yoneda lemma
- Exponential laws
    - $C^0 \cong 1$; $0^C \cong 0$ given an object in C
    - $C^1 \cong C$; $1^C \cong 1$
    - $C^{A+B} \cong C^A \times C^B$
    - $(A \times B)^C \cong A^C \times B^C$
    - $(A^B)^C \cong A^{B\times C}$
- Product laws
    - $C \times D \cong D \times C$
    - $C \times 0 \cong 0 \times C \cong 0$
    - $C \times 1 \cong 1 \times C \cong C$
- Grothendieck construction (oplax colimit) of a pseudofunctor to Cat
- Category of sections (gives rise to oplax limit of a pseudofunctor to Cat when applied to Grothendieck construction
- functor composition is functorial (there's a functor $\Delta: (C \to D) \to (D \to$

Presentation is **not** mainly about:

Presentation is **not** mainly about:

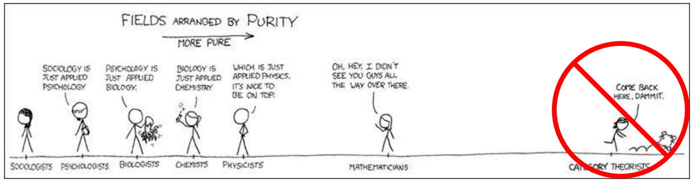
- category theory or diagram chasing



Cartoon from xkcd, adapted by Alan Huang

Presentation is **not** mainly about:

- category theory or diagram chasing

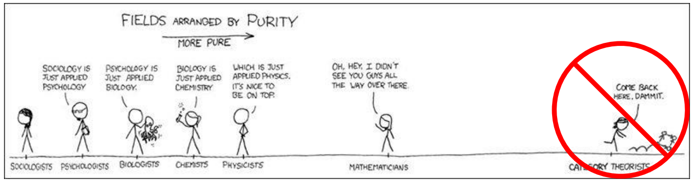


Cartoon from xkcd, adapted by Alan Huang

- the mathematical content of the library

Presentation is **not** mainly about:

- category theory or diagram chasing



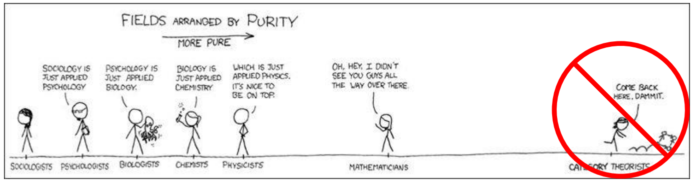
Cartoon from xkcd, adapted by Alan Huang

- the mathematical content of the library

- Coq

Presentation is **not** mainly about:

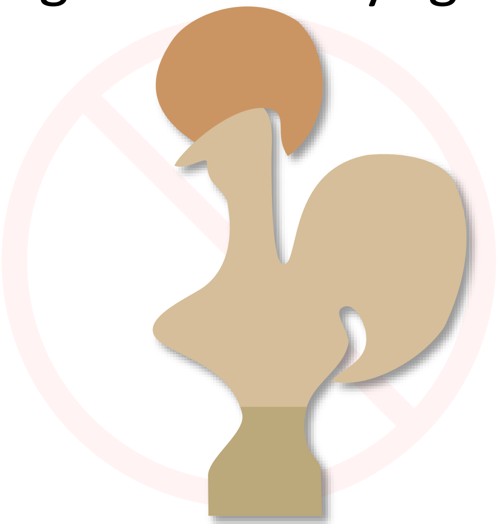
- category theory or diagram chasing



Cartoon from xkcd, adapted by Alan Huang

- the mathematical content of the library

- Coq (though what I say might not always generalize nicely)



Presentation **is** about:

- performance

- the design of proof assistants and type theories to assist with performance

- the kind of performance issues I encountered

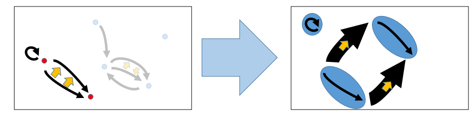
- an overview of the content of the HoTT/HoTT library

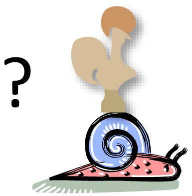
Presentation **is** for:

- Homotopy type theorists
  - Who are interested in the HoTT/HoTT library
- Users of proof assistants (and Coq in particular)
  - Who want to make their code faster
- Designers of (type-theoretic) proof assistants
  - Who want to know where to focus their optimization efforts

Outline

- Why should we care about performance?
- Overview of the HoTT/HoTT library
- What makes theorem provers (mainly Coq) slow?
  - Examples of particular slowness





- For users (workarounds)
  - Arguments vs. fields and packed records
  - Abstraction barriers

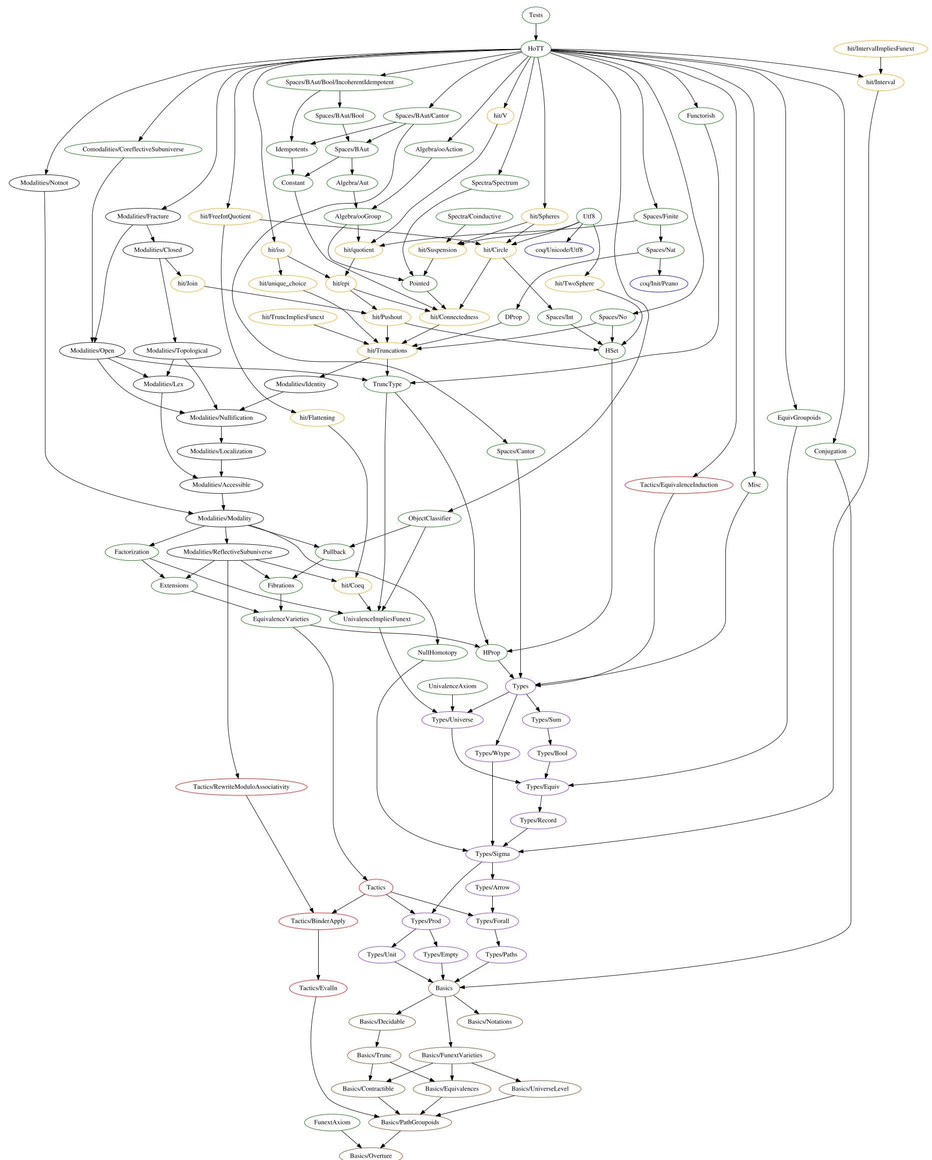
- For developers (features)
  - Primitive projections

Dam image from http://www.flickr.com/photos/gammaman/7803829282/ by Eli Christman, CC by 2.0

HoTT/HoTT Library: Contents

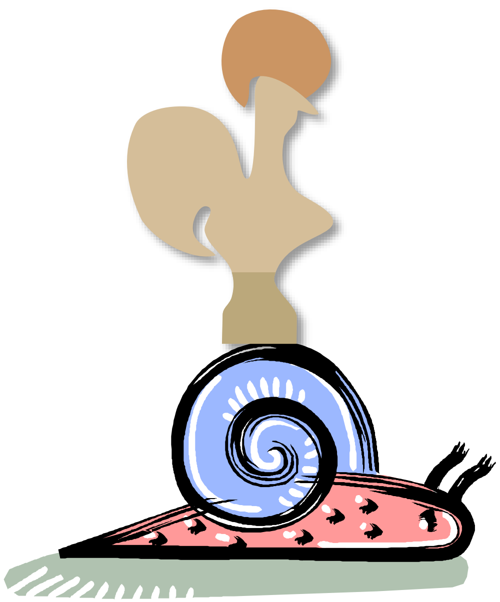
- Basic type formers and their identity types
- h-levels, object classifier, …
- Many examples of HITs from the book:
  - Circle, interval, suspensions, flattening, truncations, quotients
  - $\pi_1(S^1) = \mathbb{Z}$
- Modalities (reflective subtoposes)
- Spaces: Cantor, Finite, Surreals, …
- Categories

HoTT/HoTT Library: Diagram



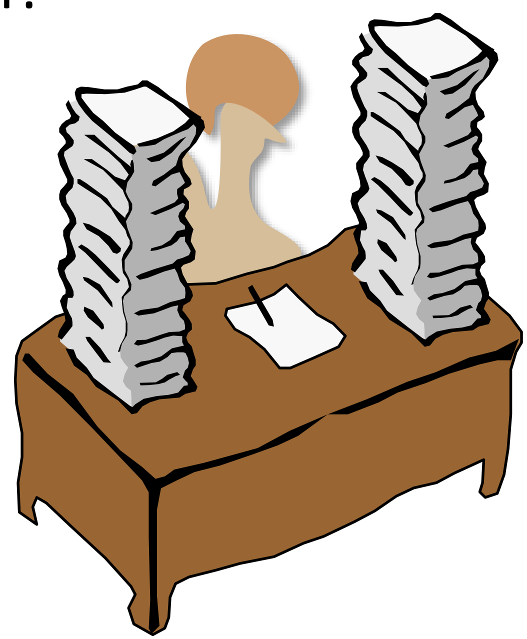
Performance

- **Question:** What makes programs, particularly theorem provers or proof scripts, slow?



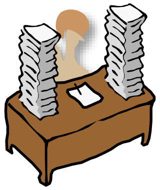
Performance

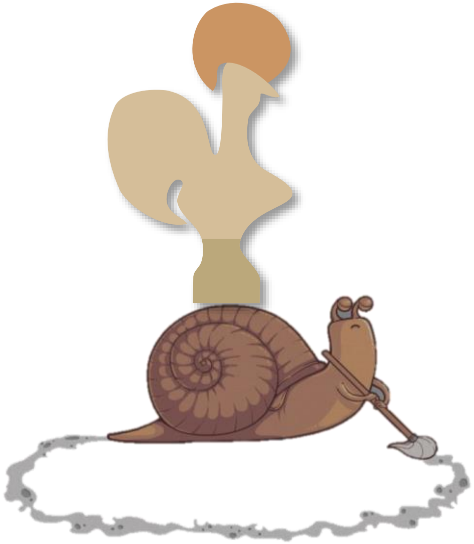
- **Question:** What makes programs, particularly theorem provers or proof scripts, slow?
- **Answer:** Doing too much stuff!



Performance

- **Question:** What makes programs, particularly theorem provers or proof scripts, slow?
- **Answer:** Doing too much stuff!
  - doing the same things repeatedly

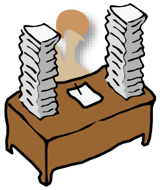




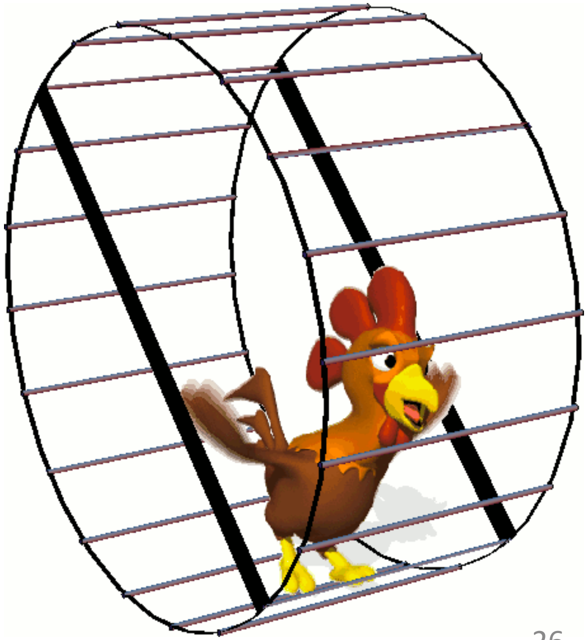
Snail from http://naolito.deviantart.com/art/Repetitive-task-258126598

Performance

- **Question:** What makes programs, particularly theorem provers or proof scripts, slow?
- **Answer:** Doing too much stuff!
  - doing the same things repeatedly



  - doing lots of stuff for no good reason



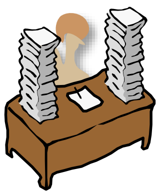
Running rooster from http://d.wapday.com:8080/animation/ccontennt/15545-f/mr_rooster_running.gif

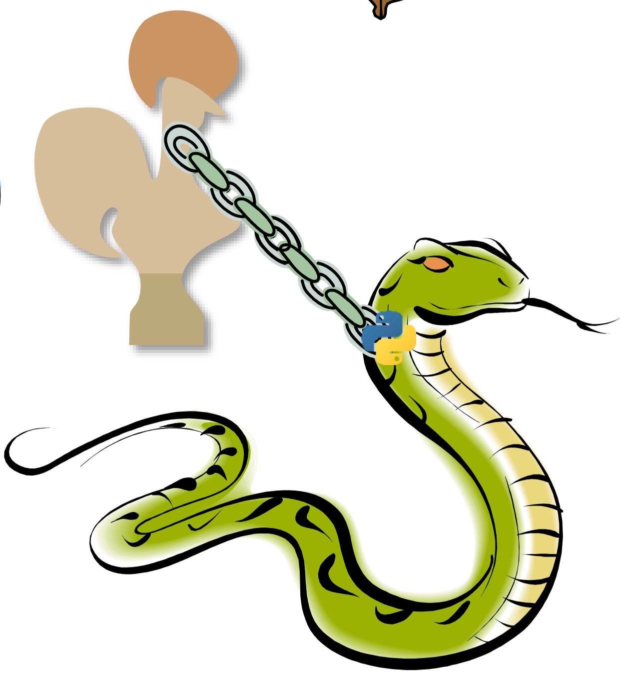
Performance

- **Question:** What makes programs, particularly theorem provers or proof scripts, slow?
- **Answer:** Doing too much stuff!
  - doing the same things repeatedly

  - doing lots of stuff for no good reason

  - using a slow language when you could be using a quicker one





Proof assistant performance

- What kinds of things does Coq do?
  - Type checking
  - Term building
  - Unification
  - Normalization

Proof assistant performance (pain)

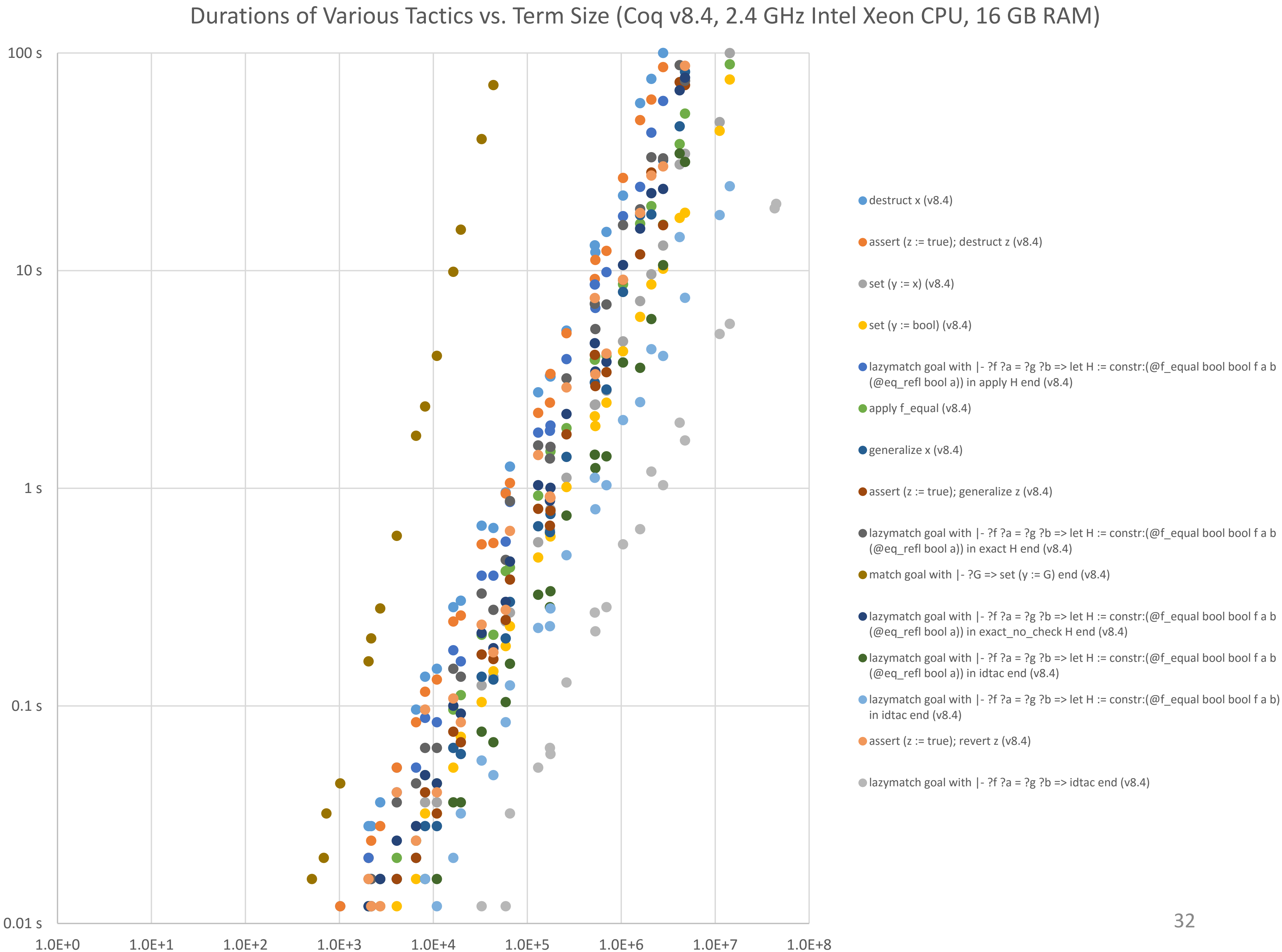
- When are these slow?
  - when you duplicate work
  - when you do work on a part of a term you end up not caring about
  - when you do them too many times
  - when your term is large

Proof assistant performance (size)

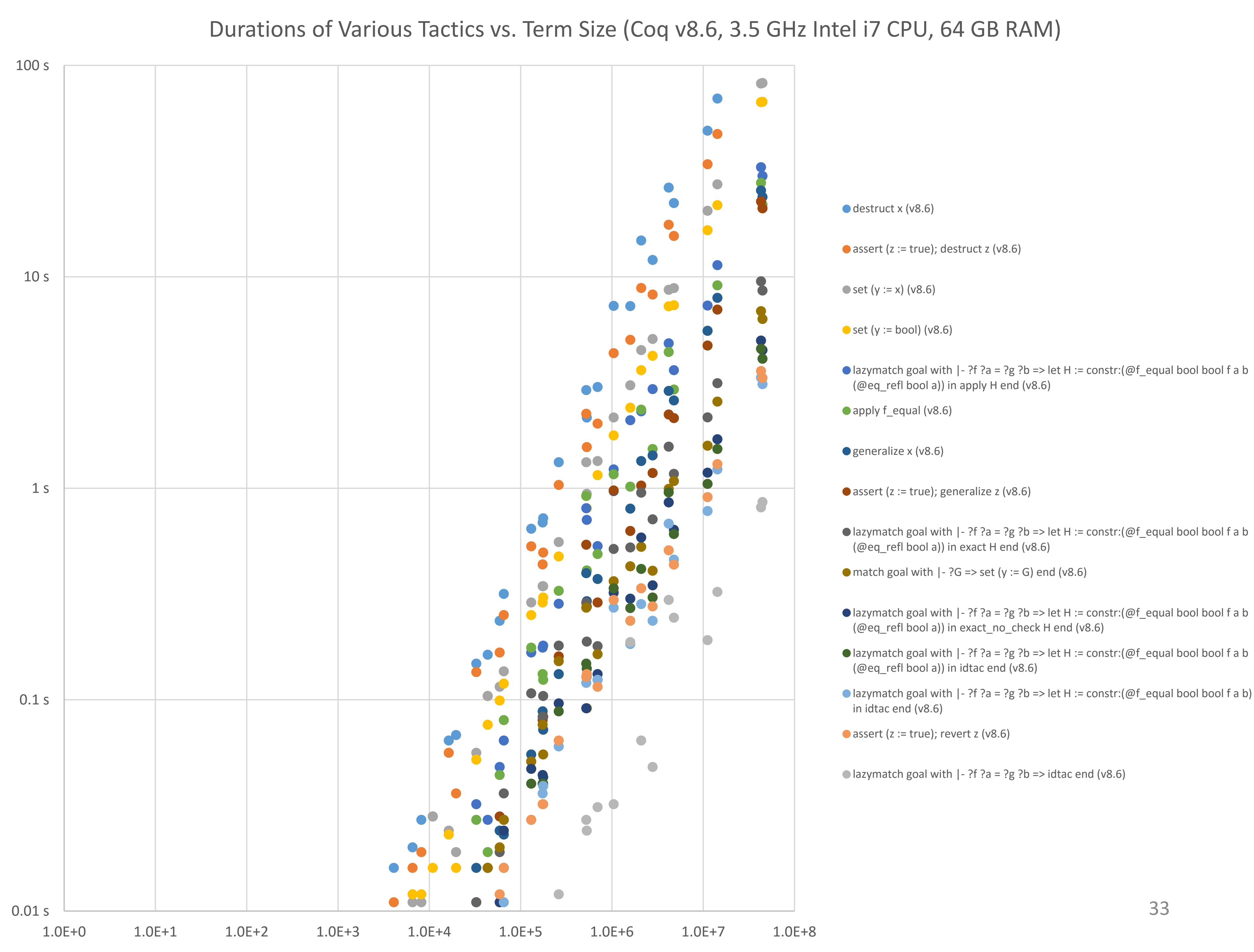
- How large is slow?

Proof assistant performance (size)

- How large is slow?
  - Around 150,000—500,000 words



Durations of Various Tactics vs. Term Size (Coq v8.4, 2.4 GHz Intel Xeon CPU, 16 GB RAM)
100 s
10 s
1 s
0.1 s
0.01 s
1.0E+0
1.0E+1
1.0E+2
1.0E+3
1.0E+4
1.0E+5
1.0E+6
1.0E+7
1.0E+8
destruct x (v8.4)
assert (z := true); destruct z (v8.4)
set (y := x) (v8.4)
set (y := bool) (v8.4)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b (@eq_refl bool a)) in apply H end (v8.4)
apply f_equal (v8.4)
generalize x (v8.4)
assert (z := true); generalize z (v8.4)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b (@eq_refl bool a)) in exact H end (v8.4)
match goal with |- ?G => set (y := G) end (v8.4)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b (@eq_refl bool a)) in exact_no_check H end (v8.4)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b (@eq_refl bool a)) in idtac end (v8.4)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b) in idtac end (v8.4)
assert (z := true); revert z (v8.4)
lazymatch goal with |- ?f ?a = ?g ?b => idtac end (v8.4)


Durations of Various Tactics vs. Term Size (Coq v8.6, 3.5 GHz Intel i7 CPU, 64 GB RAM)
100 s
10 s
1 s
0.1 s
0.01 s
1.0E+0
1.0E+1
1.0E+2
1.0E+3
1.0E+4
1.0E+5
1.0E+6
1.0E+7
1.0E+8
destruct x (v8.6)
assert (z := true); destruct z (v8.6)
set (y := x) (v8.6)
set (y := bool) (v8.6)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b (@eq_refl bool a)) in apply H end (v8.6)
apply f_equal (v8.6)
generalize x (v8.6)
assert (z := true); generalize z (v8.6)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b (@eq_refl bool a)) in exact H end (v8.6)
match goal with |- ?G => set (y := G) end (v8.6)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b (@eq_refl bool a)) in exact_no_check H end (v8.6)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b (@eq_refl bool a)) in idtac end (v8.6)
lazymatch goal with |- ?f ?a = ?g ?b => let H := constr:(@f_equal bool bool f a b) in idtac end (v8.6)
assert (z := true); revert z (v8.6)
lazymatch goal with |- ?f ?a = ?g ?b => idtac end (v8.6)

Proof assistant performance (size)

- How large is slow?
  - Around 150,000—500,000 words

Do terms actually get this large?

Proof assistant performance (size)

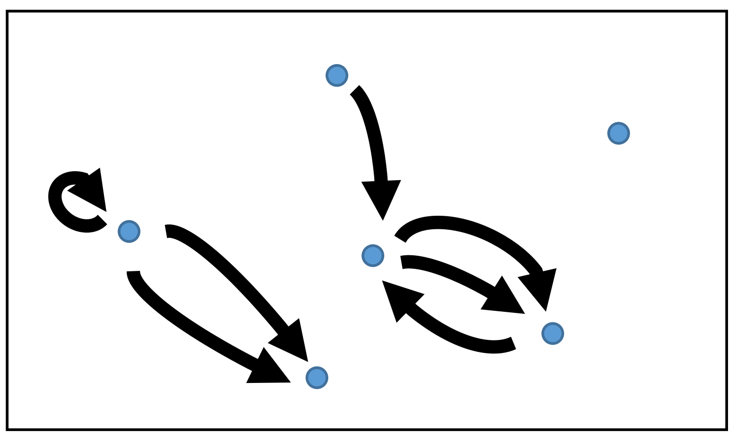
- How large is slow?
  - Around 150,000—500,000 words

Do terms actually get this large?

YES!

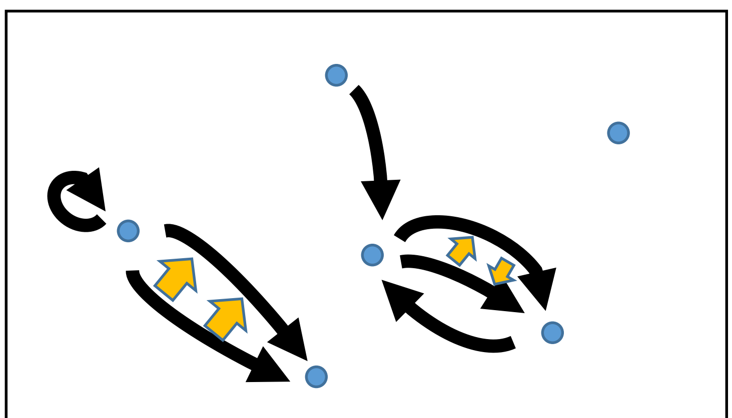
Proof assistant performance (size)

- A **directed graph** has:
  - a type of vertices (points)
  - for every ordered pair of vertices, a type of arrows



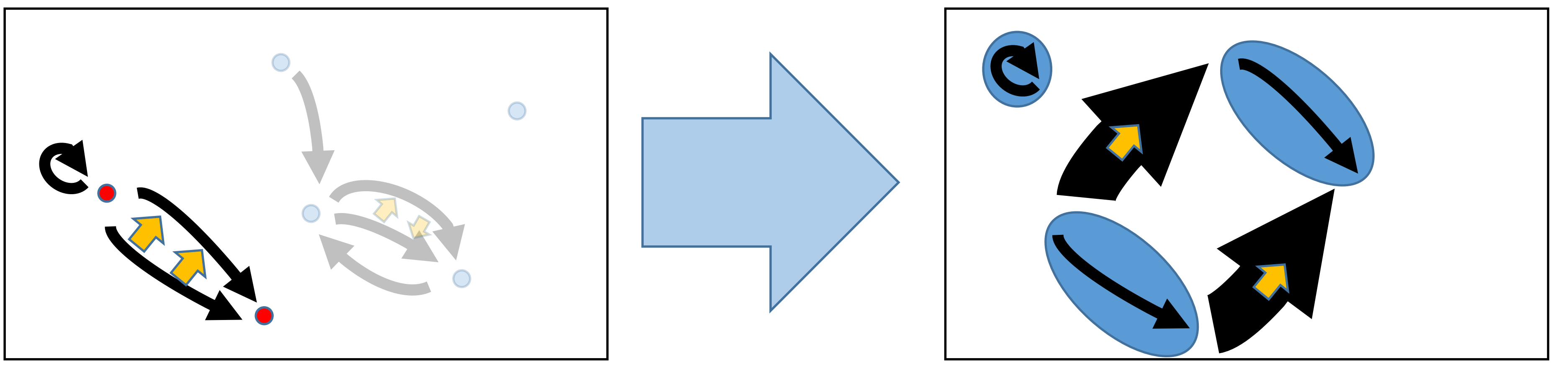
Proof assistant performance (size)

- A **directed 2-graph** has:
  - a type of vertices (0-arrows)
  - for every ordered pair of vertices, a type of arrows (1-arrows)
  - for every ordered pair of 1-arrows between the same vertices, a type of 2-arrows



Proof assistant performance (size)

- A **directed arrow-graph** comes from turning arrows into vertices:



Proof assistant performance (pain)

- When are these slow?
  - When your term is large
- Smallish example (29 000 words): Without Proofs:

$$\{|\ \mathrm{LCCM_F} \coloneqq \_\backslash\_\mathrm{induced_F}\,(m_{22} \circ m_{12});$$
$$\mathrm{LCCM_T} \coloneqq \lambda_T\,(\lambda\,(c : d'_2\ /\ F) \Rightarrow m_{21}\ c.\beta \circ m_{11}\ c.\beta)\ |\} =$$
$$\{|\ \mathrm{LCCM_F} \coloneqq \_\backslash\_\mathrm{induced_F}\ m_{12} \circ \_\backslash\_\mathrm{induced_F}\ m_{22};$$
$$\mathrm{LCCM_T} \coloneqq \lambda_T\,(\lambda\,(c : d'_2\ /\ F) \Rightarrow m_{21}\ c.\beta \circ (d_1)_1\ \mathbb{I} \circ m_{11}\ c.\beta \circ \mathbb{I})\ |\}$$

Kan Extensions

Proof assistant performance (pain)

- When are these slow?
  - When your term is large
- Smallish example (29 000 words): Without Proofs:

$$\{| \;\; \mathrm{LCCM_F} \coloneqq \_\backslash\_\mathrm{induced_F}\,(m_{22} \circ m_{12});$$
$$\mathrm{LCCM_T} \coloneqq \lambda_T\,(\lambda\,(c : d_2' \;/\; F) \Rightarrow m_{21}\,c.\beta \circ m_{11}\,c.\beta)$$
$$(\Pi{-}\mathrm{pf}\; s_2\,(\lambda_T\,(\lambda\,(c : C) \Rightarrow m_{21}\,c \circ m_{11}\,c)$$
$$(\circ_1 - \mathrm{pf}\;\; m_{21}\,m_{11}))\,(m_{22} \circ m_{12}))\; |\} =$$
$$\{| \;\; \mathrm{LCCM_F} \coloneqq \_\backslash\_\mathrm{induced_F}\,m_{12} \circ \_\backslash\_\mathrm{induced_F}\,m_{22};$$
$$\mathrm{LCCM_T} \coloneqq \lambda_T\,(\lambda\,(c : d_2' \;/\; F) \Rightarrow m_{21}\,c.\beta \circ (d_1)_1\,\mathbb{I} \circ m_{11}\,c.\beta \circ \mathbb{I})$$
$$(\circ_1 - \mathrm{pf} \quad (\lambda_T \quad (\lambda\,(c : d_2' \;/\; F) \Rightarrow m_{21}\,c.\beta)\,(\Pi{-}\mathrm{pf}$$
$$(\lambda_T\,(\lambda\,(c : d_2' \;/\; F) \Rightarrow (d_1)_1\,\mathbb{I} \circ m_{11}\,c.\beta \circ \mathbb{I})$$
$$(\circ_1 - \mathrm{pf} \quad (\lambda_T \quad (\lambda\,(c : d_2' \;/\; F) \Rightarrow (d$$
$$(\circ_0 - \mathrm{pf}\,(\lambda_T\,(\lambda\,(c : d_2 \;/\; F) \Rightarrow$$
$$(\Pi{-}\mathrm{pf}\; s_2\; m_{11}\; m_{1}$$

Kan Extensions

Proof assistant performance (pain)

- When are these slow?
  - When your term is large
- Smallish example (29 000 words): Without Proofs:

$$
\begin{aligned}
\{|\ & \mathrm{LCCM_F} \coloneqq \_\backslash\_\mathrm{induced_F}\ (m_{22} \circ m_{12}); \\
& \mathrm{LCCM_T} \coloneqq \lambda_T\ (\lambda\ (c : d_2' \ / F) \Rightarrow m_{21}\ c.\beta \circ m_{11}\ c.\beta) \\
& \qquad (\Pi\mathrm{-pf}\ s_2\ (\lambda_T\ (\lambda\ (c : C) \Rightarrow m_{21}\ c \circ m_{11}\ c) \\
& \qquad\qquad (\circ_1\ \mathrm{-pf}\ \ m_{21}\ m_{11}))\ (m_{22} \circ m_{12}))\ |\} =
\end{aligned}
$$

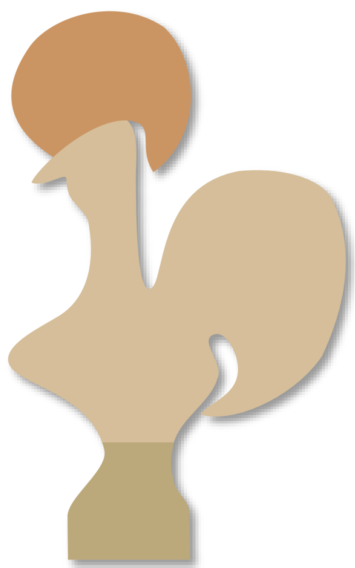
$$
\begin{aligned}
\{|\ & \mathrm{LCCM_F} \coloneqq \_\backslash\_\mathrm{induced_F}\ m_{12} \circ \_\backslash\_\mathrm{induced_F}\ m_{22}; \\
& \mathrm{LCCM_T} \coloneqq \lambda_T\ (\lambda\ (c : d_2' \ / F) \Rightarrow m_{21}\ c.\beta \circ (d_1)_1\ \mathbb{I} \circ m_{11}\ c.\beta \circ \mathbb{I}) \\
& \qquad (\circ_1\ \mathrm{-pf}\ \ (\lambda_T\ (\lambda\ (c : d_2' \ / F) \Rightarrow m_{21}\ c.\beta)\ (\Pi\mathrm{-pf}\ d_2\ m_{21}\ m_{22}))) \\
& \qquad\quad (\lambda_T\ (\lambda\ (c : d_2' \ / F) \Rightarrow (d_1)_1\ \mathbb{I} \circ m_{11}\ c.\beta \circ \mathbb{I}) \\
& \qquad\qquad (\circ_1\ \mathrm{-pf}\ (\lambda_T\ (\lambda\ (c : d_2' \ / F) \Rightarrow (d_1)_1\ \mathbb{I} \circ m_{11}\ c.\beta) \\
& \qquad\qquad\quad (\circ_0\ \mathrm{-pf}\ \ (\lambda_T\ \ (\lambda\ (c : d_2 \ / F) \Rightarrow m_{11}\ c.\beta) \\
& \qquad\qquad\qquad (\Pi\mathrm{-pf}\ s_2\ m_{11}\ m_{12}))\ \mathbb{I}))\ \mathbb{I})))\ |\}
\end{aligned}
$$

Proof assistant performance (fixes)

- How do we work around this?

Proof assistant performance (fixes)

- How do we work around this?
- By hiding from the proof checker!



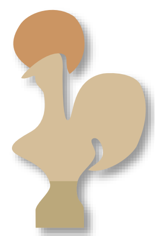
Fence from http://imgarcade.com/1/hiding-clipart/

Proof assistant performance (fixes)

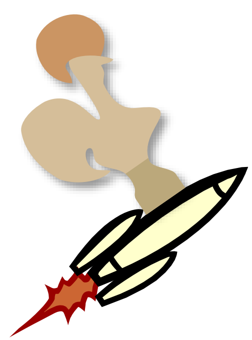
- How do we work around this?
- By hiding from the proof checker!
- How do we hide?

Proof assistant performance (fixes)

- How do we work around this?
- By hiding from the proof checker!
- How do we hide?
  - Good engineering



  - Better proof assistants

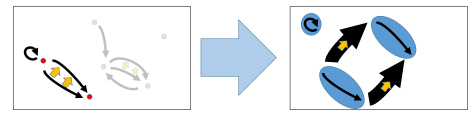


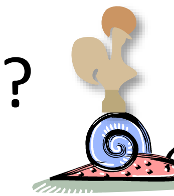
Proof assistant performance (fixes)

Careful Engineering

Outline

- Why should we care about performance?
- Overview of the HoTT/HoTT library
- What makes theorem provers (mainly Coq) slow?
  - Examples of particular slowness





- **For users (workarounds)**
  - **Arguments vs. fields and packed records**
  - **Abstraction barriers**

- For developers (features)
  - Primitive projections

Dam image from http://www.flickr.com/photos/gammaman/7803829282/ by Eli Christman, CC by 2.0

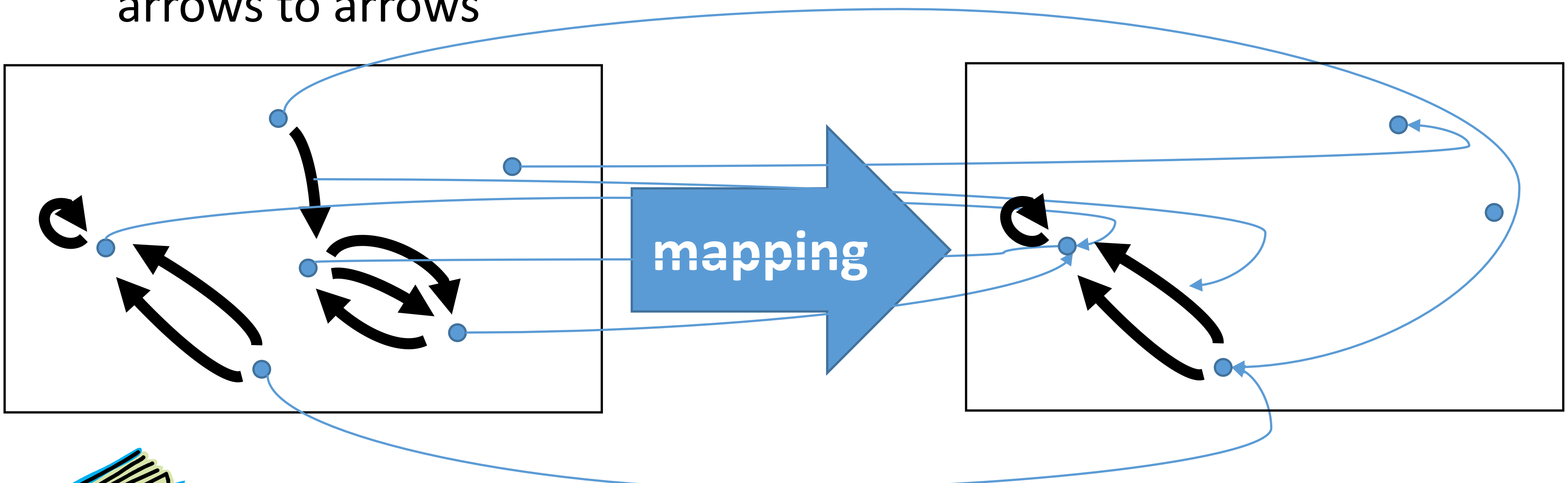
Proof assistant performance (fixes)

- How?
  - Avoid exponential blowup: Pack your records!

Proof assistant performance (fixes)

- How?
    - Avoid exponential blowup: Pack your records!

A **mapping of graphs** is a mapping of vetices to vertices and arrows to arrows



Proof assistant performance (fixes)

- How?
  - Avoid exponential blowup: Pack your records!

At least two options to define graph:

Record Graph := { V : Type ; E : V → V → Type }.

Record IsGraph (V : Type) (E : V → V → Type) := { }.

Proof assistant performance (fixes)

$\text{Record Graph} := \{\, \mathrm{V} : \text{Type} \,;\, \mathrm{E} : \mathrm{V} \to \mathrm{V} \to \text{Type} \,\}.$

$\text{Record IsGraph}\ (V : \text{Type})\ (E : V \to V \to \text{Type}) := \{\ \}.$

Big difference for size of functor:

$\text{Mapping} : \text{Graph} \to \text{Graph} \to \text{Type}.$

vs.

$$\text{IsMapping} : \forall\ (V_G : \text{Type})\ (V_H : \text{Type})\ (E_G : V_G \to V_G \to \text{Type})\ (E_H : V_H \to V_H \to \text{Type}),\ \text{IsGraph}\ V_G\ E_G \to \text{IsGraph}\ V_H\ E_H \to \text{Type}.$$

Proof assistant performance (fixes)

- How?
  - Either don’t nest constructions, or don't unfold nested constructions
  - Coq only cares about unnormalized term size – “What I don't know can't hurt me”

Proof assistant performance (fixes)

- How?
  - More systematically, have good abstraction barriers

Proof assistant performance (fixes)

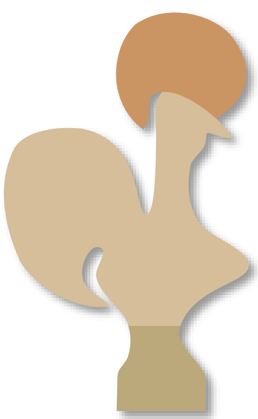
- How?
  - Have good abstraction barriers

Leaky abstraction barriers generally only torture programmers

Proof assistant performance (fixes)

- How?
  - Have good abstraction barriers

Leaky abstraction barriers torture Coq, too!



Proof assistant performance (fixes)

- How?
  - Have good abstraction barriers

Example: Pairing (without judgmental η)

Two ways to make use of elements of a pair:

$\text{let } (x, y) := p \text{ in } f\ x\ y$. (pattern matching)

$f\ (\text{fst } p)\ (\text{snd } p)$. (projections)

Proof assistant performance (fixes)

- How?
  - Have good abstraction barriers

Example: Pairing (without judgmental η)

Two ways to make use of elements of a pair:

$\text{let}\ (x, y) := p\ \text{in}\ f\ x\ y$. (pattern matching)

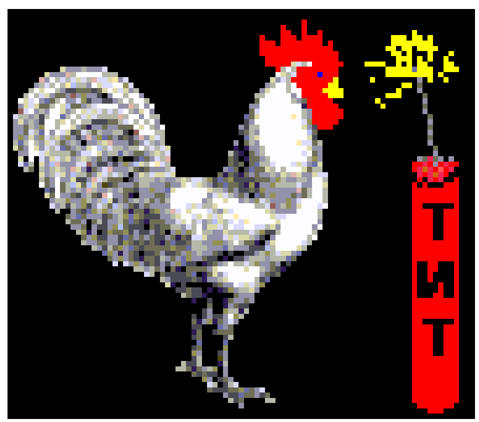
$f\ (\text{let}\ (x, y) := p\ \text{in}\ x)\ (\text{let}\ (x, y) := p\ \text{in}\ y)$. (projections)

These ways do not unify!

Proof assistant performance (fixes)

- How?
  - Have good abstraction barriers

Leaky abstraction barriers torture Coq, too!



Rooster Image from http://www.animationlibrary.com/animation/18342/Chicken_blows_up/

Dam image from http://www.flickr.com/photos/gammaman/7803829282/ by Eli Christman, CC by 2.0

Proof assistant performance (fixes)

- How?
  - Have good abstraction barriers

Leaky abstraction barriers torture Coq, too!

Proof assistant performance (fixes)

Concrete Example (Old Version)

Local Notation mor_of Y_0 Y_1 f :=
  (let η_{Y_1} := IsInitialMorphism_morphism (@HM Y_1) in
  (@center _ (IsInitialMorphism_property (@HM Y_0) _ (η_{Y_1} ∘ f))) $_1$) (only parsing).
Lemma composition_of x y z g f : mor_of _ _ (f ∘ g) = mor_of y z f ∘ mor_of x y g.
Proof.
 simpl.
 match goal with | [⊢ ((@center ?A?H) $_2$) $_1$ = _] ⇒ erewrite (@contr A H (center _; (_; _))) end. ← 8 s
 simpl; reflexivity. ← 2 s
 Grab Existential Variables.
 simpl in ∗.
 repeat match goal with | [⊢ appcontext[(?x $_2$) $_1$]] ⇒ generalize (x $_2$); intro end. ← 2.5 s
 rewrite ?composition_of. ← 0.5 s
 repeat try_associativity_quick (idtac; match goal with | [⊢ appcontext[?x $_1$]] ⇒ simpl rewrite x $_2$ end). ← 3.5 s
 rewrite ?left_identity, ?right_identity, ?associativity. ← 0.3 s
 reflexivity.
Qed. ← 20 s

Size of goal (after first simpl): 7312 words

Size of proof term: 66 264 words

Total time in file: 39 s

Proof assistant performance (fixes)

Concrete Example (New Version)

Local Notation mor_of Y_0 Y_1 f :=
(let η_{Y_1} := IsInitialMorphism_morphism (@HM Y_1) in
IsInitialMorphism_property_morphism (@HM Y_0) _ ($\eta_{Y_1} \circ f$)) (only parsing).
Lemma composition_of x y z g f : mor_of _ _ ($f \circ g$) = mor_of y z f ∘ mor_of x y g.
Proof.
simpl.
erewrite IsInitialMorphism_property_morphism_unique; [reflexivity |].
rewrite ?composition_of.
repeat try_associativity_quick rewrite IsInitialMorphism_property_morphism_property.
reflexivity.
Qed.

0.08 s (was 10 s)

0.08 s (was 0.5 s)

0.5 s (was 3.5 s)

0.5 s (was 3.5 s)

Size of goal (after first simpl): 191 words (was 7312)
Size of proof term: 3 632 words (was 66 264)
Total time in file: 3 s (was 39 s)

Proof assistant performance (fixes)

Concrete Example (Old Interface)

```
Definition IsInitialMorphism_object (M : IsInitialMorphism Aφ) : D := CommaCategory.b Aφ.
Definition IsInitialMorphism_morphism (M : IsInitialMorphism Aφ) : morphism C X (U 0 (IsInitialMorphism_object M)) := CommaCategory.f Aφ.
Definition IsInitialMorphism_property (M : IsInitialMorphism Aφ)  (Y : D) (f : morphism C X (U 0 Y))
: Contr { m : morphism D (IsInitialMorphism_object M) Y | U 1 m ∘ (IsInitialMorphism_morphism M) = f }.
Proof.
 (** We could just [rewrite right_identity], but we want to preserve judgemental computation rules. *)
 pose proof (@trunc_equiv' _ _ (symmetry _ _ (@CommaCategory.issig_morphism _ _ _ !X U _ _)) -2 (M (CommaCategory.Build_object !X U tt Y f))) as H'.
 simpl in H'.
 apply contr_inhabited_hprop.
 - abstract (
     apply @trunc_succ in H';
     eapply @trunc_equiv'; [ | exact H' ];
     match goal with
       | [ ⊢ appcontext[?m ∘ 𝕀] ] ⇒ simpl rewrite (right_identity _ _ _ m)
       | [ ⊢ appcontext[𝕀 ∘ ?m] ] ⇒ simpl rewrite (left_identity _ _ _ m)
     end;
     simpl; unfold IsInitialMorphism_object, IsInitialMorphism_morphism;
     let A := match goal with ⊢ Equiv ?A ?B ⇒ constr:(A) end in
     let B := match goal with ⊢ Equiv ?A ?B ⇒ constr:(B) end in
     apply (equiv_adjointify (λ x : A ⇒ x 2) (λ x : B ⇒ (tt; x)));
     [ intro; reflexivity | intros [[]]; reflexivity ]
   ).
 - (exists ((@center _ H') 2) 1).
   abstract (etransitivity; [ apply ((@center _ H') 2) 2 | auto with morphism ]).
Defined.
```

3 s (for the abstract block above)

1 s (for the final abstract line)

Total file time: 7 s

Proof assistant performance (fixes)

Concrete Example (New Interface)

```
Definition IsInitialMorphism_object (M : IsInitialMorphism Aφ) : D := CommaCategory.b Aφ.
Definition IsInitialMorphism_morphism (M : IsInitialMorphism Aφ) : morphism C X (U _0 (IsInitialMorphism_object M)) := CommaCategory.f Aφ.
Definition IsInitialMorphism_property_morphism (M : IsInitialMorphism Aφ)  (Y : D) (f : morphism C X (U _0 Y)) : morphism D (IsInitialMorphism_object M) Y
 := CommaCategory.h (@center _ (M (CommaCategory.Build_object !X U tt Y f))).
Definition IsInitialMorphism_property_morphism_property (M : IsInitialMorphism Aφ)  (Y : D) (f : morphism C X (U _0 Y))
: U _1 (IsInitialMorphism_property_morphism M Y f) ∘ (IsInitialMorphism_morphism M) = f
 := CommaCategory.p (@center _ (M (CommaCategory.Build_object !X U tt Y f))) @ right_identity _ _ _ _.
Definition IsInitialMorphism_property_morphism_unique (M : IsInitialMorphism Aφ)  (Y : D) (f : morphism C X (U _0 Y)) m' (H : U _1 m' ∘ IsInitialMorphism_morphism M = f)
: IsInitialMorphism_property_morphism M Y f = m'
:= ap (@CommaCategory.h _ _ _ _ _ _ _)
      (@contr _ (M (CommaCategory.Build_object !X U tt Y f)) (CommaCategory.Build_morphism Aφ (CommaCategory.Build_object !X U tt Y f) tt m' (H @ (right_identity _ _ _ _) ^-1))).
Definition IsInitialMorphism_property (M : IsInitialMorphism Aφ)  (Y : D) (f : morphism C X (U _0 Y))
: Contr { m : morphism D (IsInitialMorphism_object M) Y | U _1 m ∘ (IsInitialMorphism_morphism M) = f }.
 := {| center := (IsInitialMorphism_property_morphism M Y f; IsInitialMorphism_property_morphism_property M Y f);
       contr m' := path_sigma _ (IsInitialMorphism_property_morphism M Y f; IsInitialMorphism_property_morphism_property M Y f)
                              m' (@ IsInitialMorphism_property_morphism_unique M Y f m' _1 m' _2) (center _) |}.
```

0.4 s

Total file time: 7 s

Proof assistant performance (fixes)

Concrete Example 2 (Generalization)

Lemma pseudofunctor_to_cat_assoc_helper $\{x\ x_0 : C\}$ $\{x_2 : \text{morphism}\ C\ \text{x}\ \text{x0}\}$ $\{\text{x1} : C\}$

$\{x_5 : \text{morphism}\ C\ x_0\ x_1\}$ $\{x_4 : C\}$ $\{x_7 : \text{morphism}\ C\ x_1\ x_4\}$

$\{p\ p_0 : \text{PreCategory}\}$ $\{f : \text{morphism}\ C\ x\ x_4 \to \text{Functor}\ p_0\ p\}$

$\{p_1\ p_2 : \text{PreCategory}\}$ $\{f_0 : \text{Functor}\ p_2\ p\}$ $\{f_1 : \text{Functor}\ p_1\ p_2\}$ $\{f_2 : \text{Functor}\ p_0\ p_2\}$ $\{f_3 : \text{Functor}\ p_0\ p_1\}$ $\{f_4 : \text{Functor}\ p_1\ p\}$

$\{x_{16} : \text{morphism}\ (\_ \to \_)\ (f\ (x_7 \circ x_5 \circ x_2))\ (f_4 \circ f_3)\%\text{functor}\}$

$\{x_{15} : \text{morphism}\ (\_ \to \_)\ f_2\ (f_1 \circ f_3)\%\text{functor}\}$ $\{H_2 : \text{IsIsomorphism}\ x_{15}\}$

$\{x_{11} : \text{morphism}\ (\_ \to \_)\ (f\ (x_7 \circ (x_5 \circ x_2)))\ (f_0 \circ f_2)\%\text{functor}\}$

$\{H_1 : \text{IsIsomorphism}\ x_{11}\}$ $\{x_9 : \text{morphism}\ (\_ \to \_)\ f_4\ (f_0 \circ f_1)\%\text{functor}\}$ $\{\text{fst\_hyp} : x_7 \circ x_5 \circ x_2 = x_7 \circ (x_5 \circ x_2)\}$

$(\text{rew\_hyp} : \forall\ x_3 : p_0,$

$(\text{idtoiso}\ (p_0 \to p)\ (\text{ap}\ f\ \text{fst\_hyp}) : \text{morphism}\_\_\_)\ x_3 = x_{11}{}^{-1}\ x_3 \circ (f_0\ {}_1\ (x_{15}{}^{-1}\ x_3) \circ (\mathbb{I} \circ (x_9\ (f_3\ x_3) \circ x_{16}\ x_3))))$

$\{H_0' : \text{IsIsomorphism}\ x_{16}\}$ $\{H_1' : \text{IsIsomorphism}\ x_9\}$ $\{x_{13} : p\}$ $\{x_3 : p_0\}$ $\{x_6 : p_1\}$ $\{x_{10} : p_2\}$

$\{x_{14} : \text{morphism}\ p\ (f_0\ x_{10})\ x_{13}\}$ $\{x_{12} : \text{morphism}\ p_2\ (f_1\ x_6)\ x_{10}\}$ $\{x_8 : \text{morphism}\ p_1\ (f_3\ x_3)\ x_6\}$

: existT $(\lambda\ f_5 : \text{morphism}\ C\ x\ x_4 \Rightarrow \text{morphism}\ p\ ((f\ f_5)\ x_3)\ x_{13})$

$(x_7 \circ x_5 \circ x_2)$

$(x_{14} \circ (f_0\ {}_1\ x_{12} \circ x_9\ x_6) \circ (f_4\ {}_1\ x_8 \circ x_{16}\ x_3)) = (x_7 \circ (x_5 \circ x_2); x_{14} \circ (f_0\ {}_1\ (x_{12} \circ (f_1\ {}_1\ x_8 \circ x_{15}\ x_3)) \circ x_{11}\ x_3)).$

Proof.

helper_t assoc_before_commutes_tac.

assoc_fin_tac.

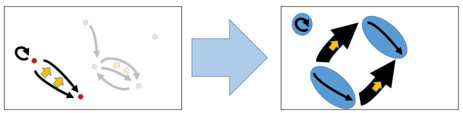
Qed.

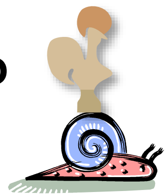
Speedup: 10x for the file, from 4m 53s to 28 s

Time spent: a few hours

Outline

- Why should we care about performance?
- Overview of the HoTT/HoTT library
- What makes theorem provers (mainly Coq) slow?
  - Examples of particular slowness





- **For users (workarounds)**
  - **Arguments vs. fields and packed records**
  - **Abstraction barriers**

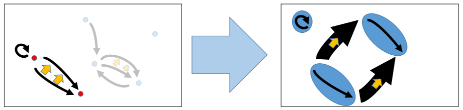
- For developers (features)
  - Primitive projections

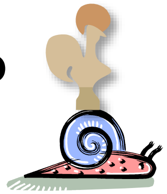
Proof assistant performance (fixes)

Better Proof Assistants

Outline

- Why should we care about performance?
- Overview of the HoTT/HoTT library
- What makes theorem provers (mainly Coq) slow?
  - Examples of particular slowness





- For users (workarounds)
  - Arguments vs. fields and packed records
  - Abstraction barriers

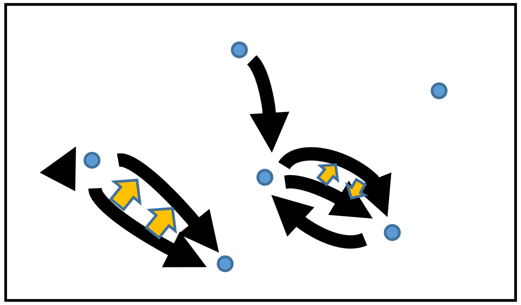
- **For developers (features)**
  - **Primitive projections**

Proof assistant performance (fixes)

- How?
  - Primitive projections

Proof assistant performance (fixes)

- How?
  - Primitive projections



Definition 2-Graph :=

{ V : Type &

{ 1E : V → V → Type &

∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type }}.

Definition V (G : 2-Graph) := pr_1 G .

Definition 1E (G : 2-Graph) := pr_1 (pr_2 G).

Definition 2E (G : 2-Graph) := pr_2 (pr_2 G).

Proof assistant performance (fixes)

Definition 2-Graph :=

{ V : Type &

{ 1E : V → V → Type &

$\forall\ v_1\ v_2$, 1E $v_1\ v_2$ → 1E $v_1\ v_2$ → Type }}.

Definition V (G : 2-Graph) := pr_1 G .

Proof assistant performance (fixes)

Definition 2-Graph :=

{ V : Type &

{ 1E : V → V → Type &

∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type }.

Definition V (G : 2-Graph) :=

@pr$_1$ Type (λ V : Type ⇒

{ 1E : V → V → Type &

∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type })

G.

Proof assistant performance (fixes)

Definition 2-Graph :=

{ V : Type &

{ 1E : V → V → Type &

∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type }}.

Definition V (G : 2-Graph) := pr_1 G .

Definition 1E (G : 2-Graph) := pr_1 (pr_2 G).

Proof assistant performance (fixes)

Definition 1E (G : 2-Graph) :=
@pr_1
(@pr_1 Type (λ V : Type ⇒
{ 1E : V → V → Type &
∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type })
G →
@pr_1 Type (λ V : Type ⇒
{ 1E : V → V → Type &
∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type })
G →
Type)
(λ 1E : @pr_1 Type (λ V : Type ⇒
{ 1E : V → V → Type &

Proof assistant performance (fixes)

```
Definition 1E (G : 2-Graph) :=
 @pr_1
   (@pr_1 Type (λ V : Type ⇒
                  { 1E : V → V → Type &
                        ∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type })
           G →
    @pr_1 Type (λ V : Type ⇒
                  { 1E : V → V → Type &
                        ∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type })
           G →
    Type)
   (λ 1E : @pr_1 Type (λ V : Type ⇒
                          { 1E : V → V → Type &
                                ∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type })
                 G →
           @pr_1 Type (λ V : Type ⇒
                          { 1E : V → V → Type &
                                ∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type })
                 G →
           Type ⇒
       ∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type)
   (@pr_2 Type (λ V : Type ⇒
                  { 1E   : V → V → Type &
                          ∀ v_1 v_2, 1E v_1 v_2 → 1E v_1 v_2 → Type }
           G)
```

Proof assistant performance (fixes)

Definition 1E (G : 2-Graph) :=
@pr_1
(@pr_1 Type (λ V : Type ⇒ { 1E : V → V → Type & ∀ (v_1 : V) (v_2 : V), 1E v_1 v_2 → 1E v_1 v_2 → Type }) G →
@pr_1 Type (λ V : Type ⇒ { 1E : V → V → Type & ∀ (v_1 : V) (v_2 : V), 1E v_1 v_2 → 1E v_1 v_2 → Type }) G →
Type)
(λ 1E : @pr_1 Type (λ V : Type ⇒ { 1E : V → V → Type & ∀ (v_1 : V) (v_2 : V), 1E v_1 v_2 → 1E v_1 v_2 → Type }) G →
@pr_1 Type (λ V : Type ⇒ { 1E : V → V → Type & ∀ (v_1 : V) (v_2 : V), 1E v_1 v_2 → 1E v_1 v_2 → Type }) G →
Type ⇒
∀(v_1 : @pr_1 Type (λ V : Type ⇒ { 1E : V → V → Type & ∀ (v_1 : V) (v_2 : V), 1E v_1 v_2 → 1E v_1 v_2 → Type }) G)
(v_2 : @pr_1 Type (λ V : Type ⇒ { 1E : V → V → Type & ∀ (v_1 : V) (v_2 : V), 1E v_1 v_2 → 1E v_1 v_2 → Type }) G),
1E v_1 v_2 → 1E v_1 v_2 → Type)
(@pr_2 Type (λ V : Type ⇒ { 1E : V → V → Type & ∀ (v_1 : V) (v_2 : V), 1E v_1 v_2 → 1E v_1 v_2 → Type }) G)
:@pr_1 Type (λ V : Type ⇒ { 1E : V → V → Type & ∀ (v_1 : V) (v_2 : V), 1E v_1 v_2 → 1E v_1 v_2 → Type }) G →
@pr_1 Type (λ V : Type ⇒ { 1E : V → V → Type & ∀ (v_1 : V) (v_2 : V), 1E v_1 v_2 → 1E v_1 v_2 → Type }) G →
Type

Recall: Original was:

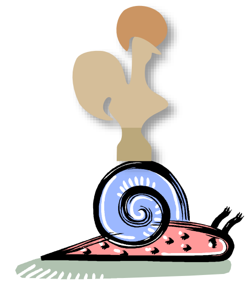
Definition 1E (G : 2-Graph) := pr_1 (pr_2 G).

Proof assistant performance (fixes)

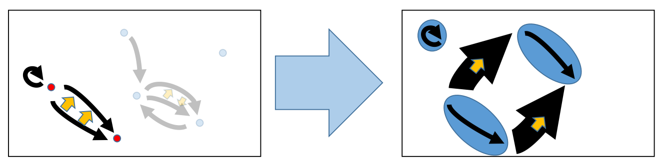
- How?
  - Primitive projections
  - They eliminate the unnecessary arguments to projections, cutting down the work Coq has to do.

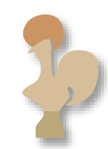
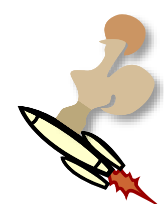
Take-away messages

- Performance matters (even in proof assistants)



- Term size matters for performance



- Performance can be improved by
  - careful engineering of developments  
  - improving the proof assistant or the metatheory  

Thank You!

The presentation will be available at

http://people.csail.mit.edu/jgross/#hott-hott-and-category-coq-experience

An extended version is available at

http://people.csail.mit.edu/jgross/#category-coq-experience

The library is available at

https://github.com/HoTT/HoTT

Questions?