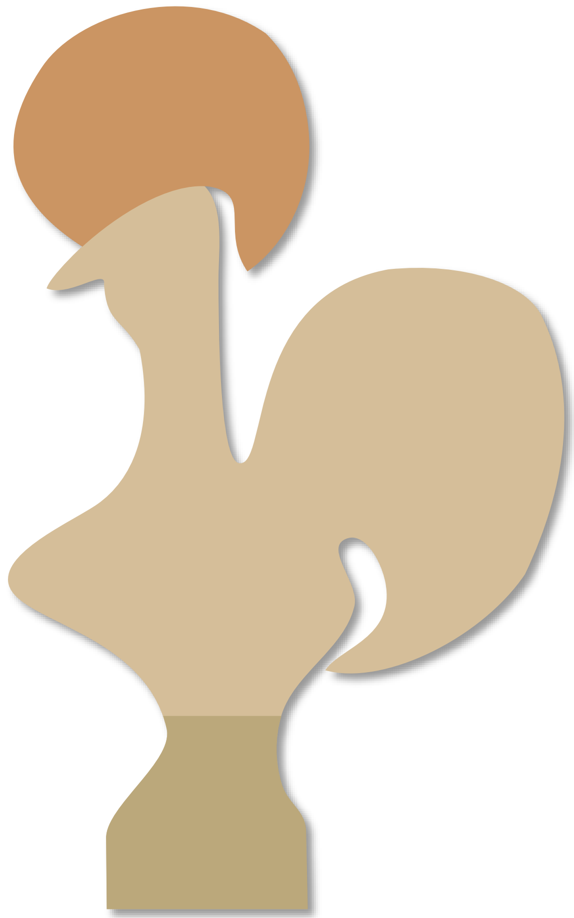
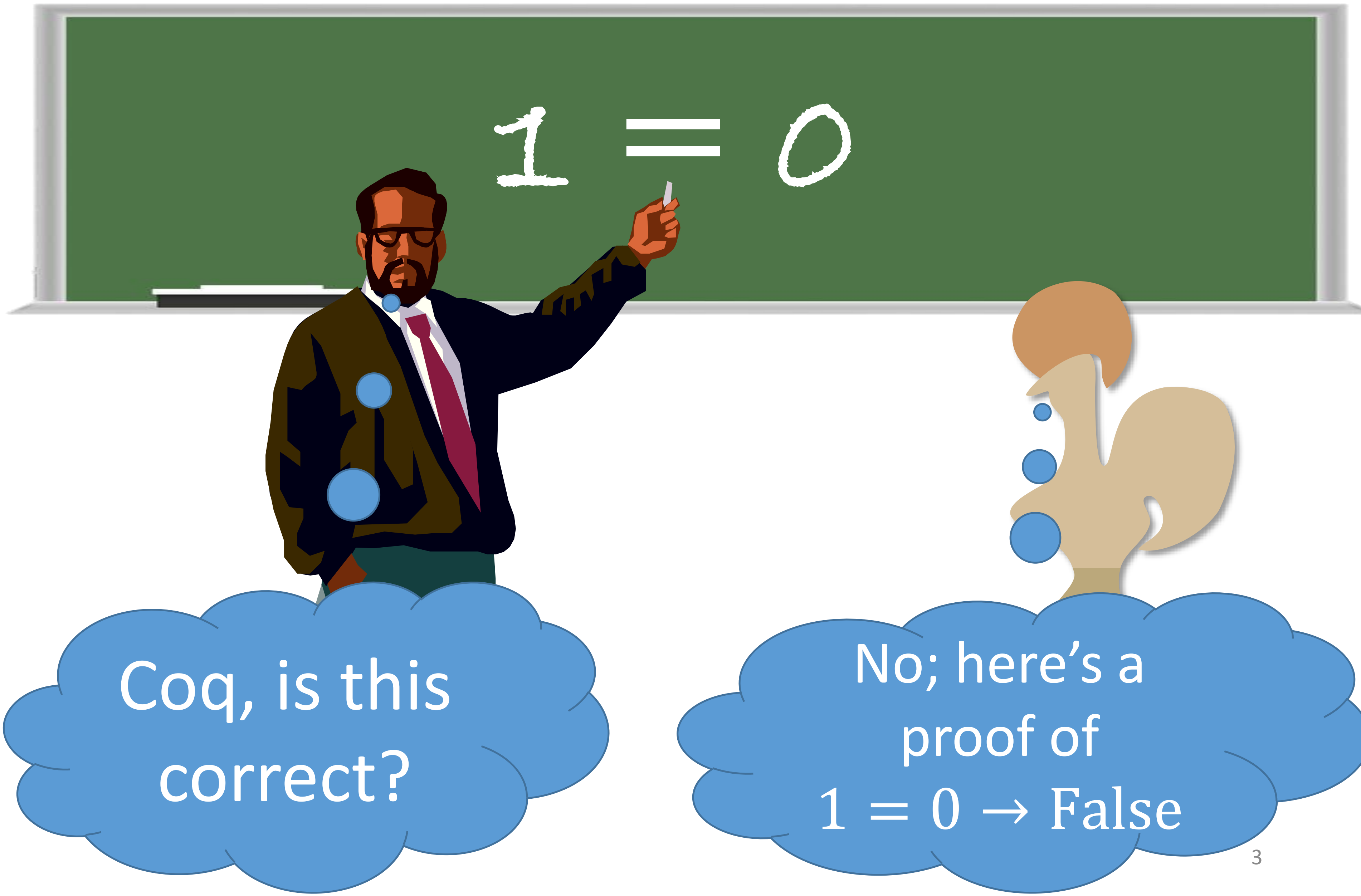
Experience implementing a performant category-theory library in Coq

Jason Gross, Adam Chlipala, David I. Spivak
Massachusetts Institute of Technology

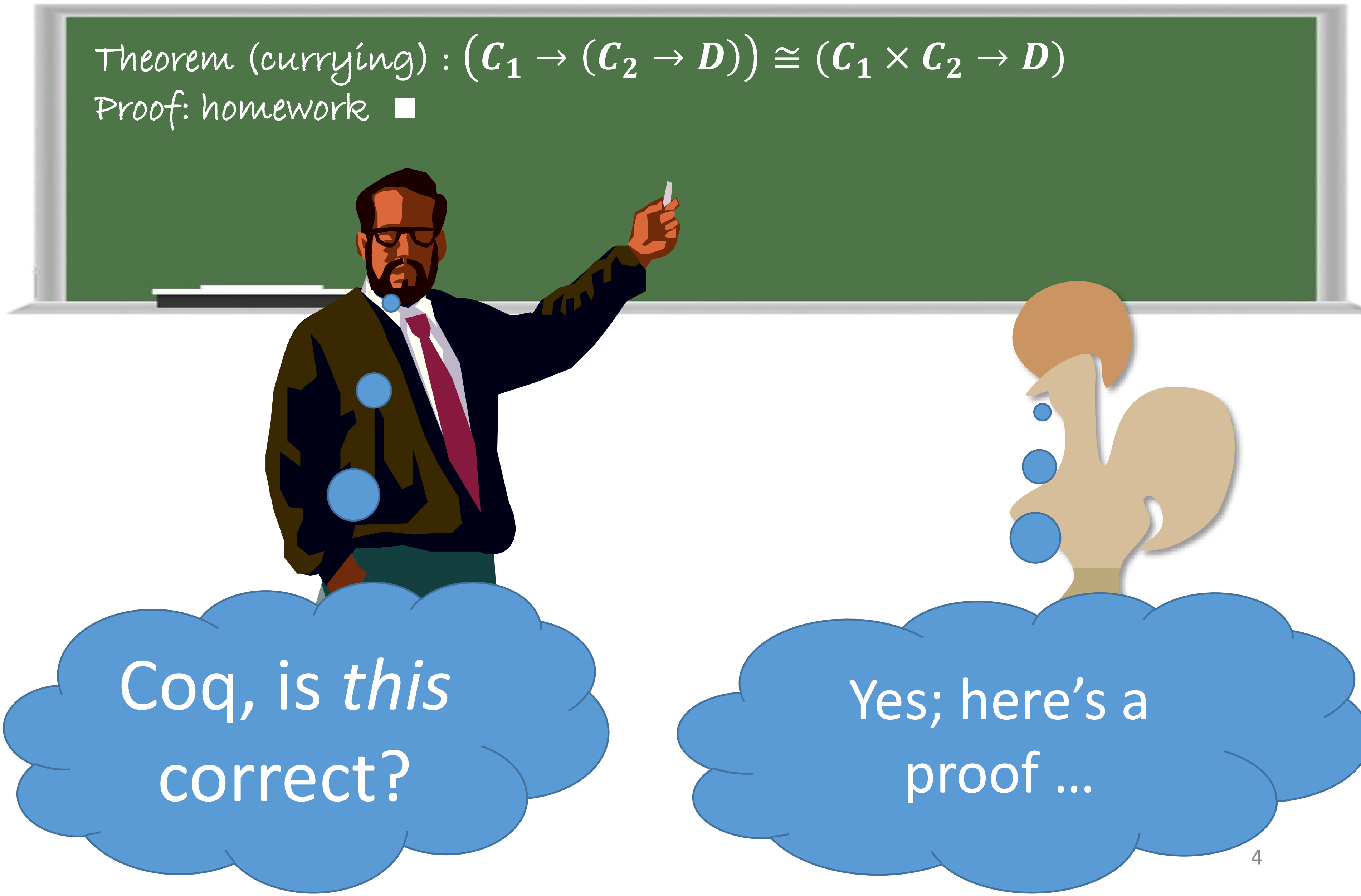
How should theorem provers work?



How theorem provers should work:



How theorem provers should work:



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: homework $\blacksquare$

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

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. $\blacksquare$

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

Proof.

esplit.

{ by refine ($\lambda_{\mathrm{F}}\ (F \mapsto (\lambda_{\mathrm{F}}\ (c \mapsto F_{\mathrm{o}}\ c_1\ c_2)))$). }

{ by refine ($\lambda_{\mathrm{F}}\ (F \mapsto (\lambda_{\mathrm{F}}\ (c_1 \mapsto (\lambda_{\mathrm{F}}\ (c_2 \mapsto F_{\mathrm{o}}\ (c_1, c_2))))))$). }

all: trivial.

Qed.

How theorem provers should work:

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

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

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

Functoriality, naturality, and congruence: straightforward. $\blacksquare$

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) : $(C_1 \to (C_2 \to D)) \cong (C_1 \times C_2 \to D)$

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

Proof.

esplit.

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

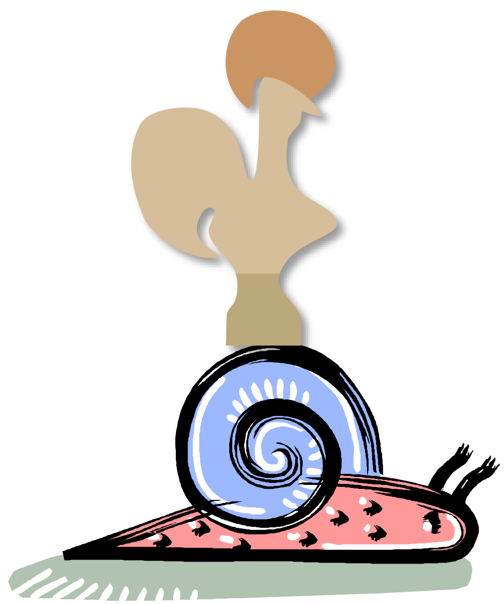
{ by refine $(\lambda_{\mathrm{F}}\ (F \mapsto (\lambda_{\mathrm{F}}\ (c_1 \mapsto (\lambda_{\mathrm{F}}\ (c_2 \mapsto F_{\mathrm{o}}\ (c_1, c_2))\ (s\ d\ m \mapsto F_{\mathrm{m}}\ (1, m))))$
$(F\ G\ T \mapsto (\lambda_{\mathrm{T}}\ (c_1 \mapsto (\lambda_{\mathrm{T}}\ (c_2 \mapsto T\ (c_1, c_2))))))$.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

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):

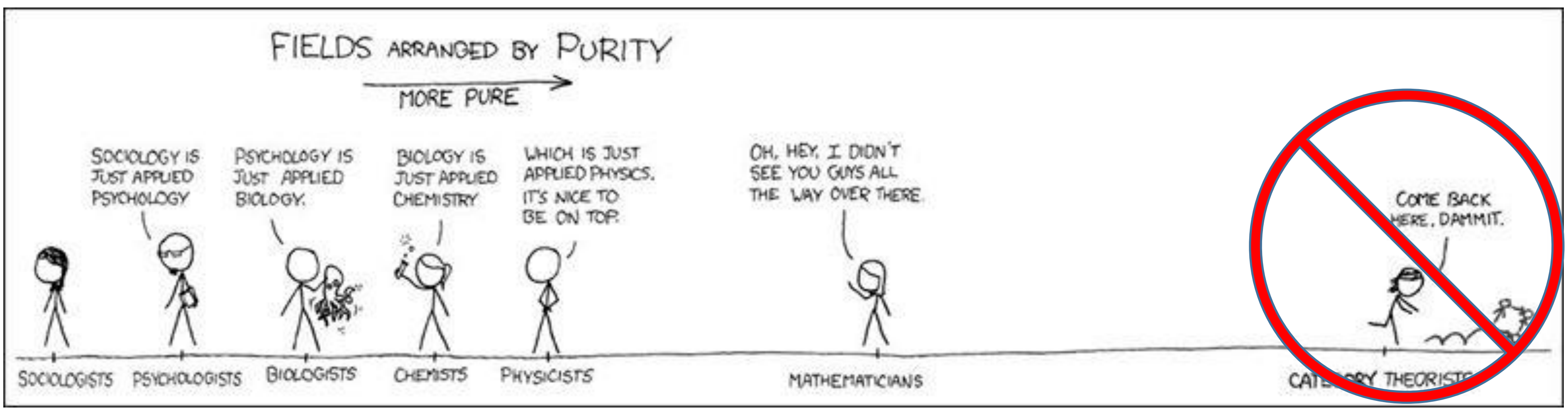
Concepts Formalized:

- 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
- profunctors
    - identity profunction (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 (porting from old version in progress)
    - 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 (still too slow to be merged into the library proper; code here)
- 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:

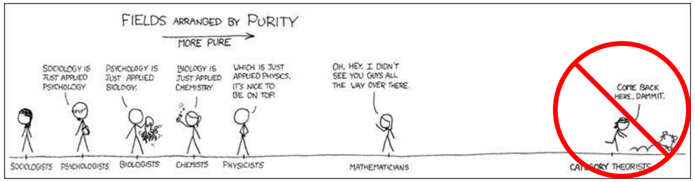
- category theory or diagram chasing



Cartoon from xkcd, adapted by Alan Huang

Presentation is **not** mainly about:

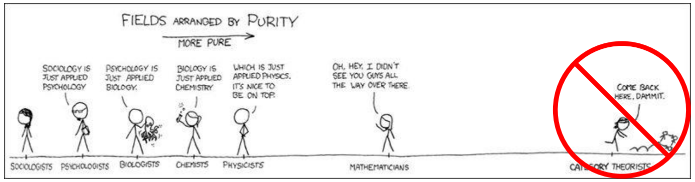
- category theory or diagram chasing



Cartoon from xkcd, adapted by Alan Huang

- my library

Presentation is **not** mainly about:



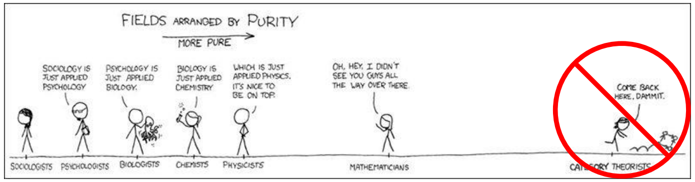
Cartoon from xkcd, adapted by Alan Huang

- category theory or diagram chasing
- my library

- Coq

Presentation is **not** mainly about:

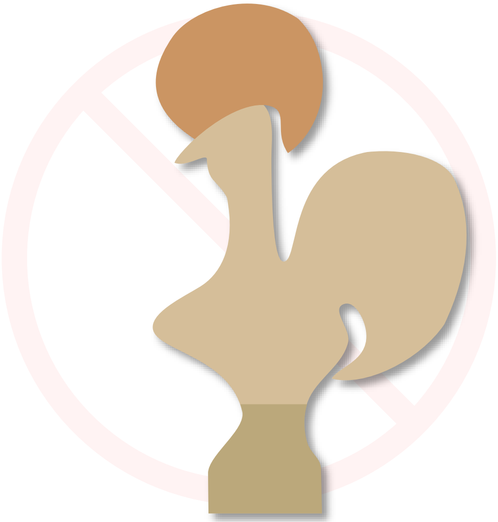
- category theory or diagram chasing



Cartoon from xkcd, adapted by Alan Huang

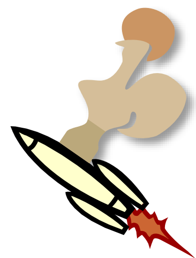
- my library

- 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

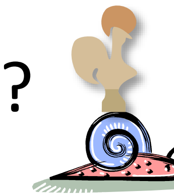
Presentation **is** for:

- Users of proof assistants (and Coq in particular)
  - Who want to make their code faster

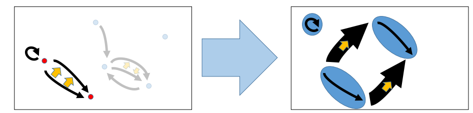
- Designers of (type-theoretic) proof assistants
  - Who want to know where to focus their optimization efforts

Outline

- Why should we care about performance?
- What makes theorem provers (mainly Coq) slow?



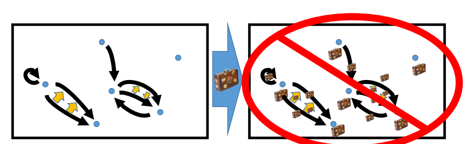
  - Examples of particular slowness



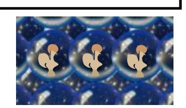
- For users (workarounds)
  - Arguments vs. fields and packed records
  - Proof by duality as proof by unification
  - Abstraction barriers

  - Proof by reflection

- For developers (features)
  - Primitive projections
  - Higher inductive types



  - Universe Polymorphism



  - More judgmental rules
  - Hashconsing

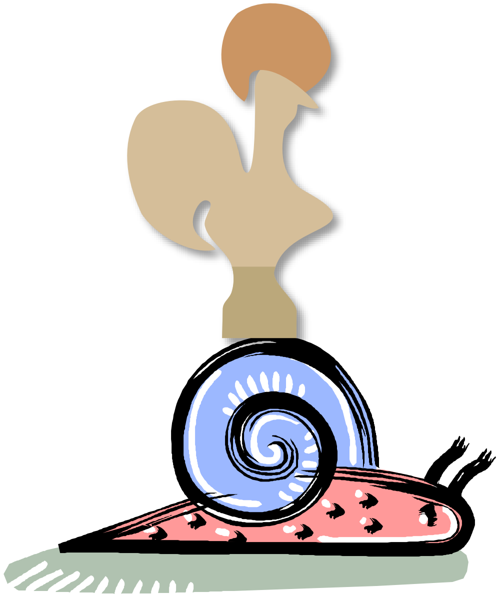
Universes image from Abell NGC2218 hst big, NASA, http://en.wikipedia.org/wiki/Abell_2218#mediaviewer/File:Abell_NGC2218_hst_big.jpg, released in Public Domain; Bubble from http://pixabay.com/en/blue-bubble-shiny-157652/, released in Public Domain CC0, combined in Photoshop by Jason Gross

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

Fence image from http://www.picgifs.com/clip-art/playing-children/clip-art-playing-children-362018-689955/

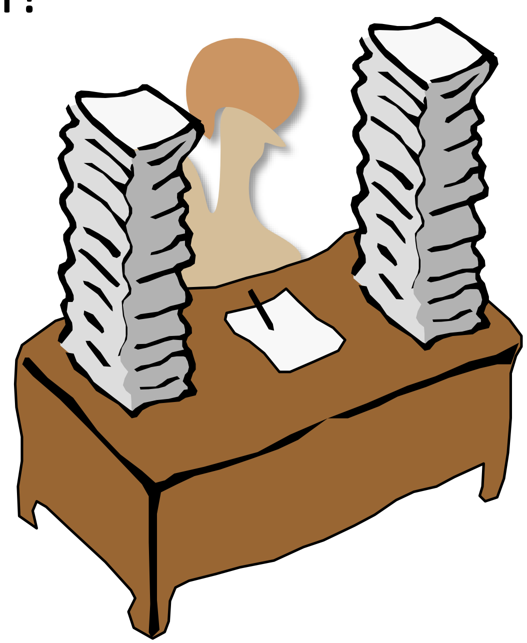
Performance

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



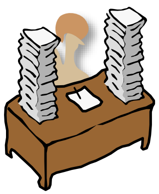
Performance

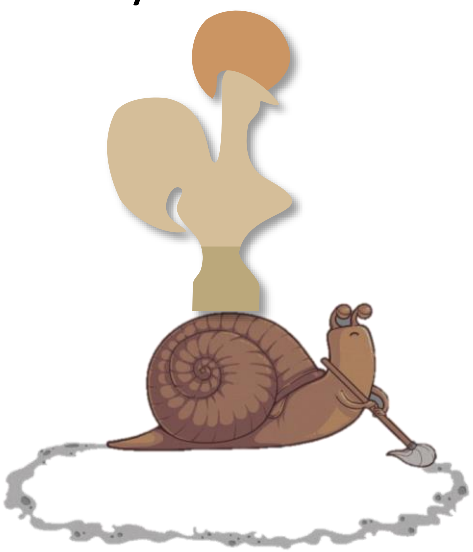
- **Question:** What makes programs, particularly theorem provers or proof scripts, slow?
- **Answer:** Doing too much stuff!



Performance

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

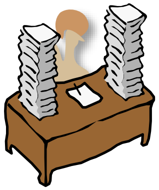




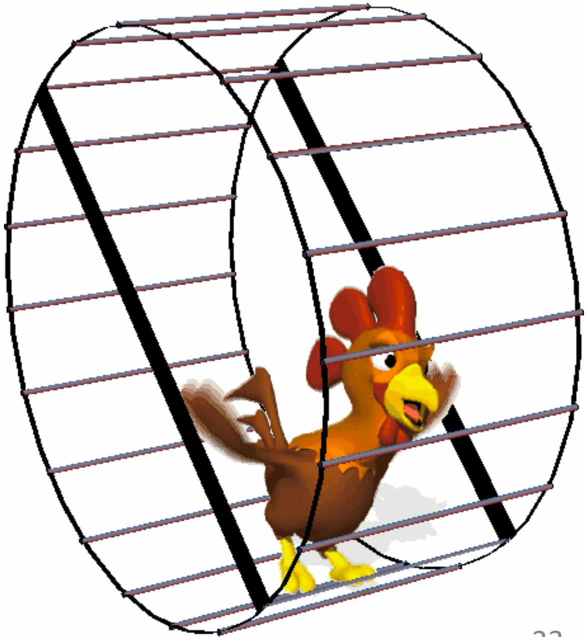
Snail from http://naolito.deviantart.com/art/Repetitive-task-258126598

Performance

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



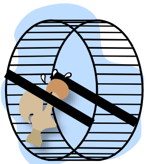
  - doing lots of stuff for no good reason

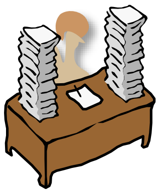


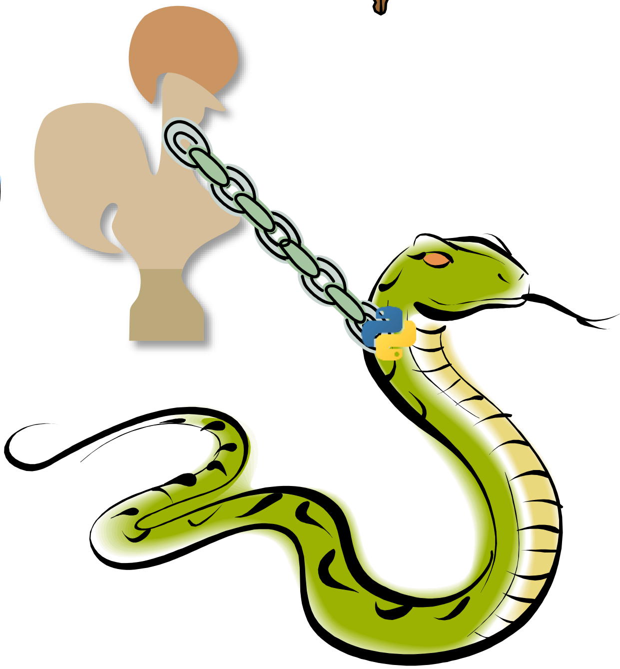
Running rooster from http://d.wapday.com:8080/animation/ccontennt/15545-f/mr_rooster_running.gif

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)

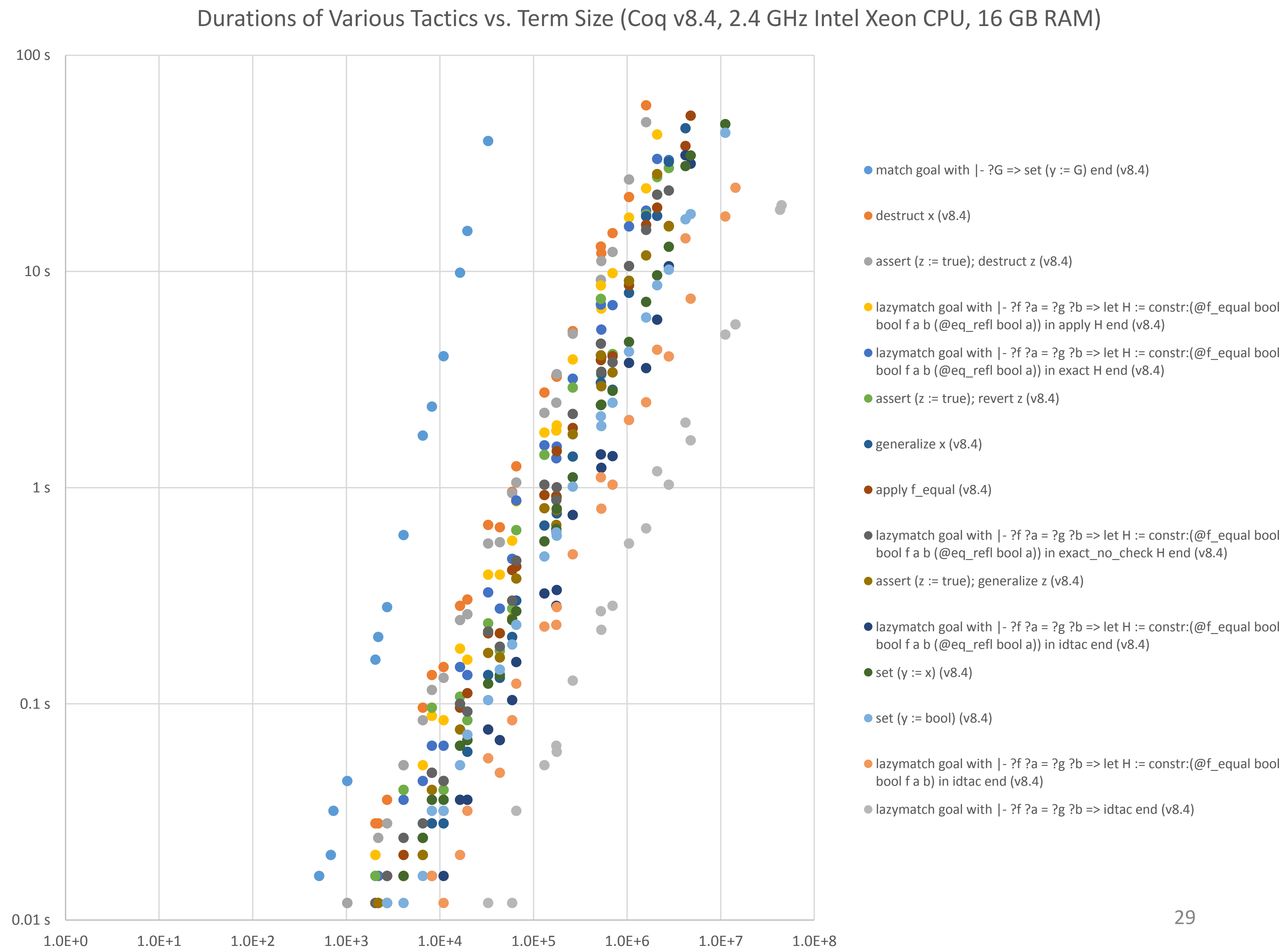
- When are these slow?
  - when you duplicate work
  - when you do work on a part of a term you end up not caring about
  - when you do them too many times
  - when your term is large

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
match goal with |- ?G => set (y := G) end (v8.4)
destruct x (v8.4)
assert (z := true); destruct 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 apply 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 exact H end (v8.4)
assert (z := true); revert z (v8.4)
generalize x (v8.4)
apply f_equal (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)
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 idtac end (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) in idtac end (v8.4)
lazymatch goal with |- ?f ?a = ?g ?b => idtac end (v8.4)

Proof assistant performance (size)

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

Do terms actually get this large?

Proof assistant performance (size)

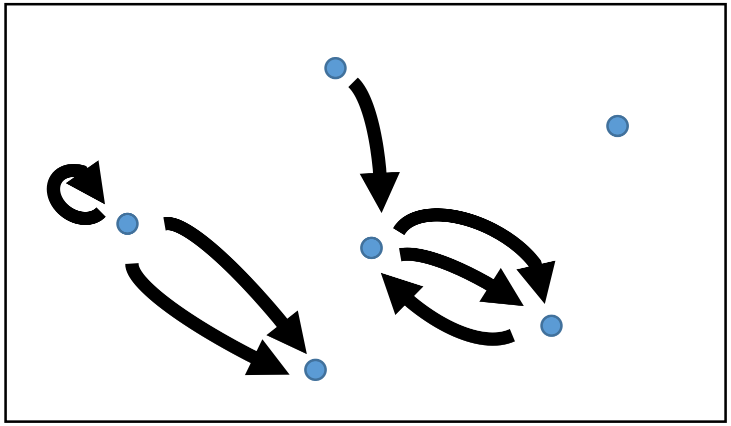
- How large is slow?
  - Around 150,000—500,000 words

Do terms actually get this large?

YES!

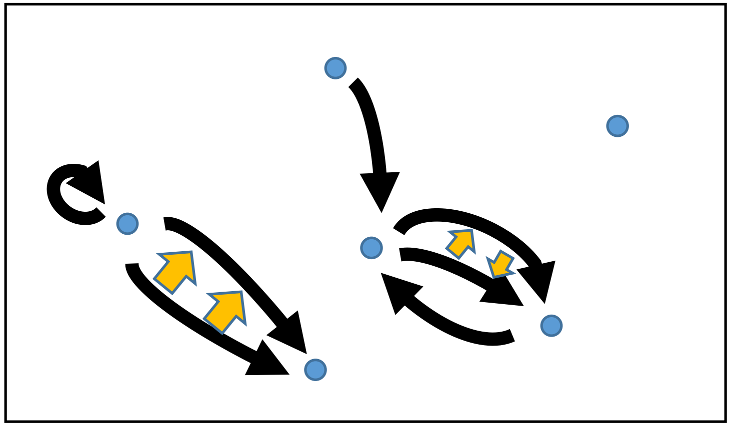
Proof assistant performance (size)

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



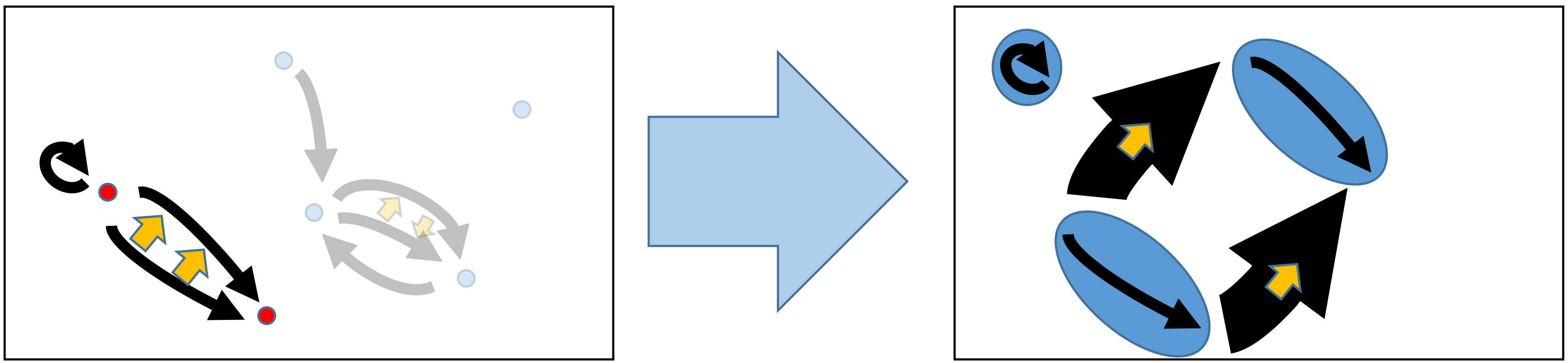
Proof assistant performance (size)

- 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:

$$
\begin{aligned}
\{|\ \ & \mathrm{LCCM}_{\mathrm{F}} \coloneqq \_\backslash\_\mathrm{induced}_{\mathrm{F}}\,(m_{22} \circ m_{12});\\
& \mathrm{LCCM}_{\mathrm{T}} \coloneqq \lambda_T\,(\lambda\,(c : d'_2\ \ /\ F) \Rightarrow m_{21}\ c.\beta \circ m_{11}\ c.\beta)\ |\} =\\
\{|\ \ & \mathrm{LCCM}_{\mathrm{F}} \coloneqq \_\backslash\_\mathrm{induced}_{\mathrm{F}}\,m_{12} \circ \_\backslash\_\mathrm{induced}_{\mathrm{F}}\,m_{22};\\
& \mathrm{LCCM}_{\mathrm{T}} \coloneqq \lambda_T\,(\lambda\,(c : d'_2\ \ /\ F) \Rightarrow m_{21}\ c.\beta \circ (d_1)_1\ \mathbb{I} \circ m_{11}\ c.\beta \circ \mathbb{I})\ |\}
\end{aligned}
$$

Proof assistant performance (pain)

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

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

Proof assistant performance (pain)

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

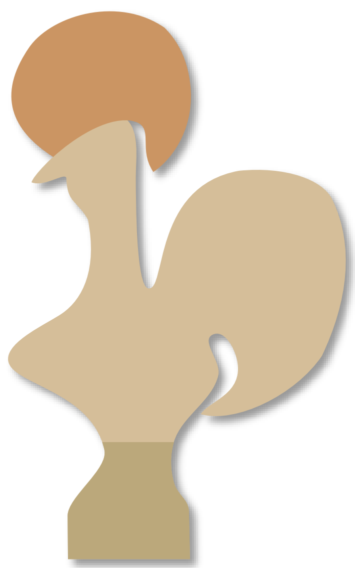
$$
\begin{aligned}
\{|\ & \mathrm{LCCM_F} \coloneqq \_\backslash\_\mathrm{induced_F}\ (m_{22} \circ m_{12});\\
& \mathrm{LCCM_T} \coloneqq \lambda_T\ (\lambda\ (c : d_2'\ \ / F) \Rightarrow m_{21}\ c.\beta \circ m_{11}\ c.\beta)\\
&\qquad\qquad (\Pi\!-\!\mathrm{pf}\ s_2\ (\lambda_T\ (\lambda\ (c : C)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

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!

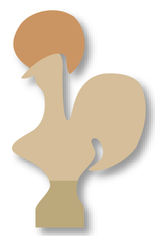
Fence from http://imgarcade.com/1/hiding-clipart/

Proof assistant performance (fixes)

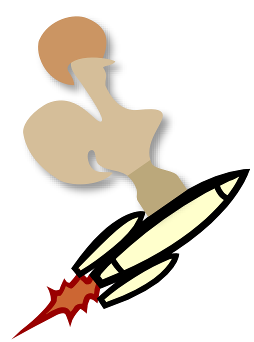
- How do we work around this?
- By hiding from the proof checker!
- How do we hide?

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

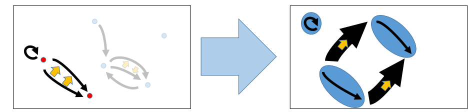


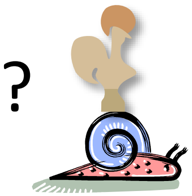
Proof assistant performance (fixes)

Careful Engineering

Outline

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





- **For users (workarounds)**
  - **Arguments vs. fields and packed records**
  - **Proof by duality as proof by unification**
  - **Abstraction barriers**
  - **Proof by reflection**

- For developers (features)
  - Primitive projections
  - Higher inductive types
  - Universe Polymorphism
  - More judgmental rules
  - Hashconsing

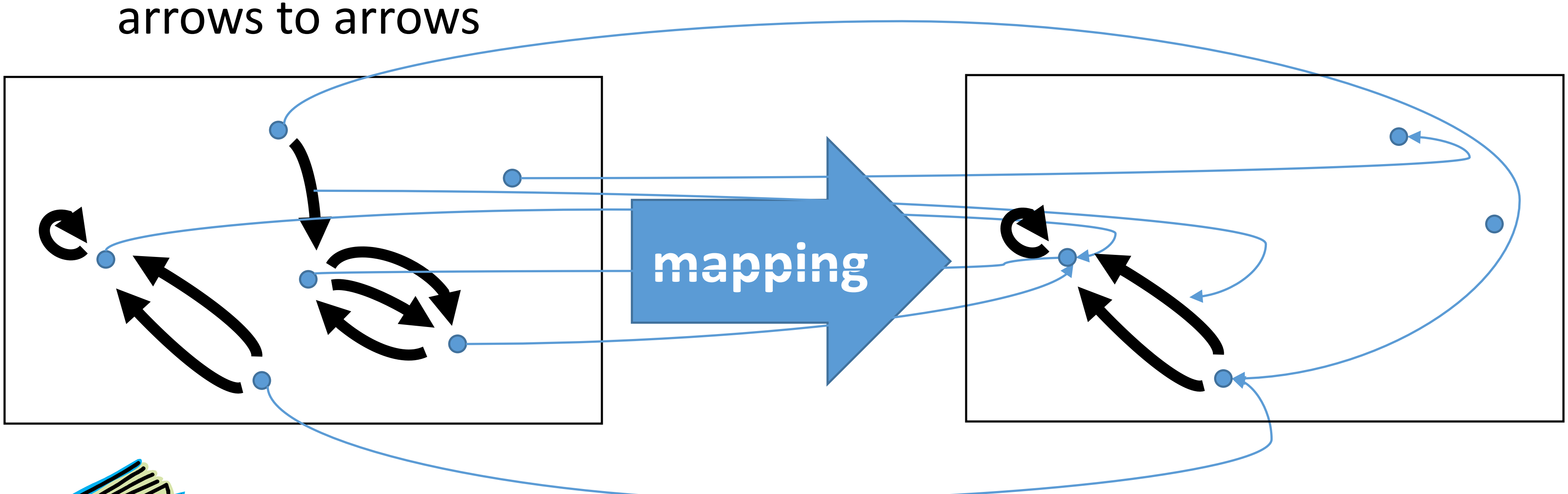
Proof assistant performance (fixes)

- How?
  - Pack your records!

Proof assistant performance (fixes)

- How?
  - Pack your records!

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



Proof assistant performance (fixes)

- How?
  - 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.

$$\begin{aligned}\text{IsMapping} : \forall\ & (V_G : \text{Type})\ (V_H : \text{Type}) \\ & (E_G : V_G \to V_G \to \text{Type})\ (E_H : V_H \to V_H \to \text{Type}), \\ & \quad \text{IsGraph}\ V_G\ E_G \to \text{IsGraph}\ V_H\ E_H \to \text{Type}.\end{aligned}$$

Proof assistant performance (fixes)

- How?
  - Exceedingly careful engineering to get proofs for free

Proof assistant performance (fixes)

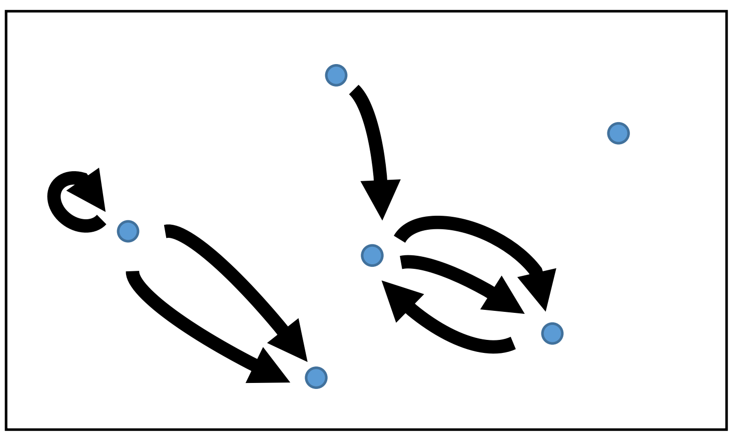
- Duality proofs for free

Proof assistant performance (fixes)

- Duality proofs for free
- Idea: One proof, two theorems

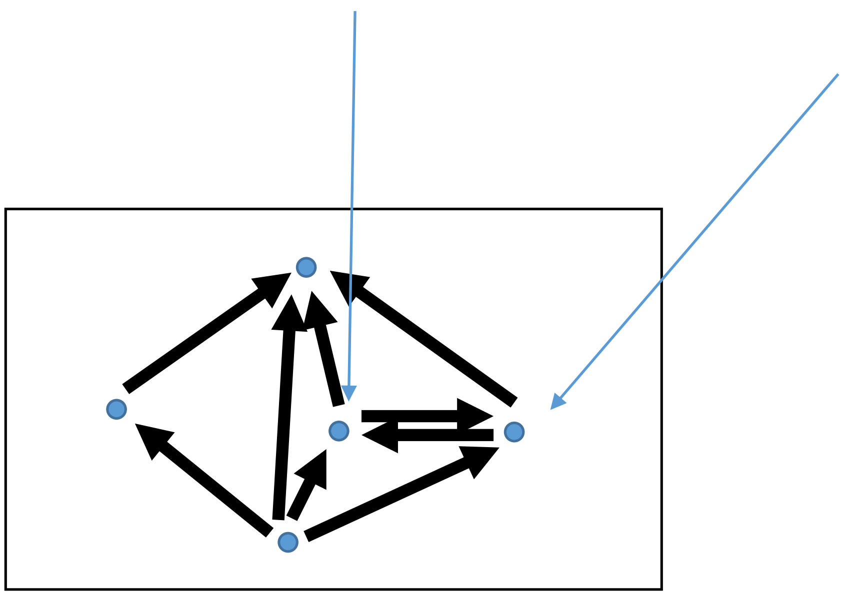
Proof assistant performance (fixes)

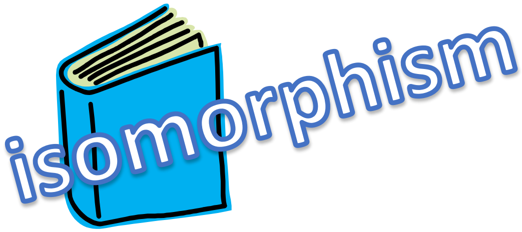
- Duality proofs for free
- Recall: A **directed graph** has:
  - a type of vertices (points)
  - for every ordered pair of vertices, a type of arrows



Proof assistant performance (fixes)

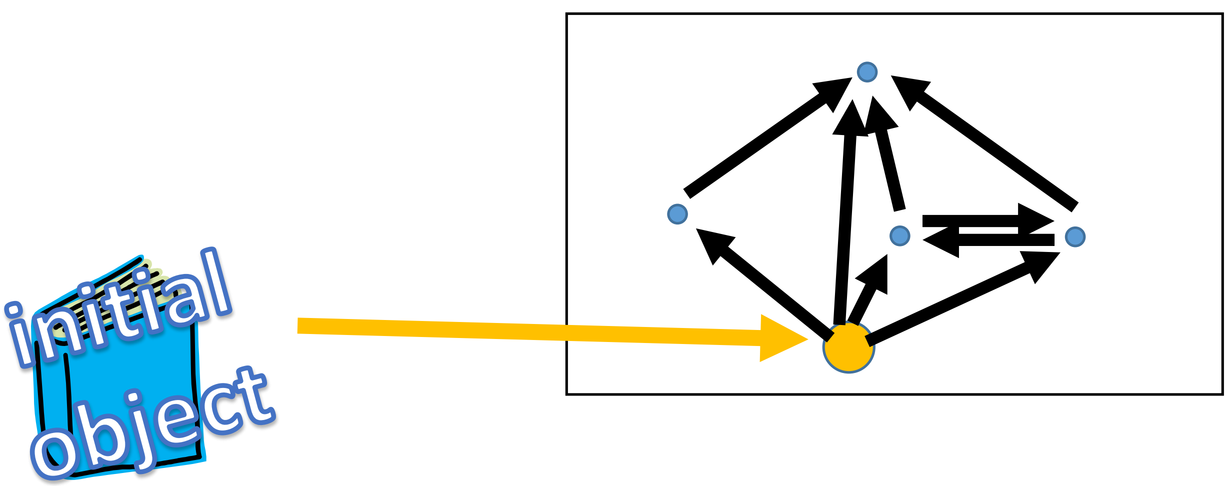
- Duality proofs for free
- Two vertices are **isomorphic** if there is exactly one edge between them in each direction





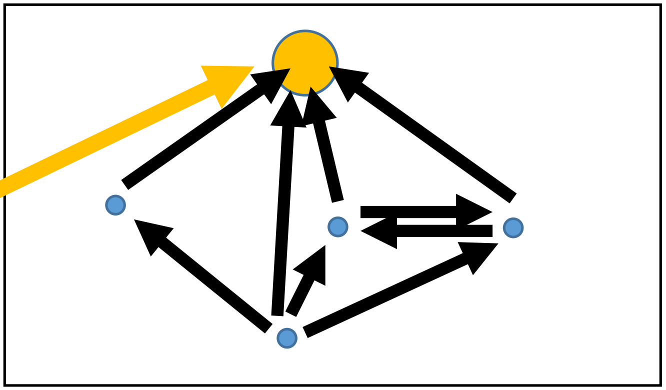
Proof assistant performance (fixes)

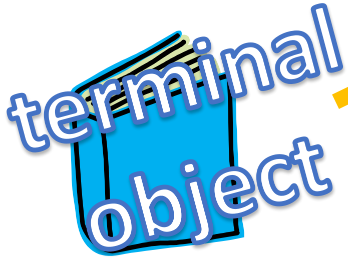
- Duality proofs for free
- Two vertices are **isomorphic** if there is exactly one edge between them in each direction
- An **initial (bottom) vertex** is a vertex with exactly one edge **to** every other vertex



Proof assistant performance (fixes)

- Duality proofs for free
- Two vertices are **isomorphic** if there is exactly one edge between them in each direction
- An **initial (bottom) vertex** is a vertex with exactly one edge **to** every other vertex
- A **terminal (top) vertex** is a vertex with exactly one edge **from** every other vertex





Proof assistant performance (fixes)

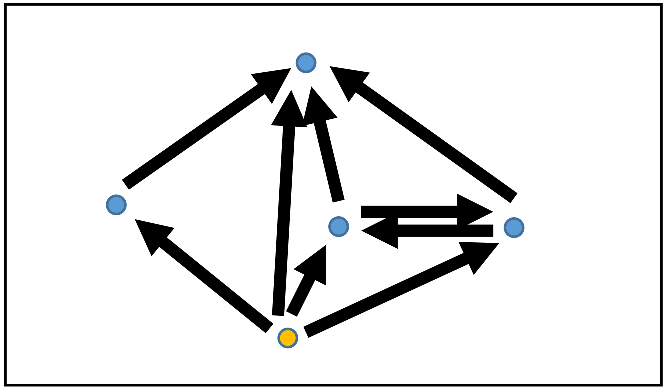
- Theorem: Initial vertices are unique

Theorem initial_unique : $\forall$ (G : Graph) (x y : G.V),

is_initial x → is_initial y → $x \cong y$

- Proof:

 Exercise for the audience



Proof assistant performance (fixes)

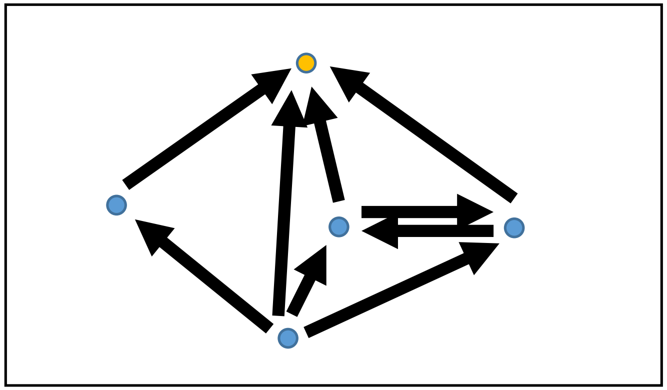
- Theorem: Terminal vertices are unique

Theorem terminal_unique : ∀ (G : Graph) (x y : G.V),

is_terminal x → is_terminal y → $x \cong y$

- Proof:

$\lambda\, G\ x\ y\ H\ H' \Rightarrow$ initial_unique $G^{\mathrm{op}}\ y\ x\ H'\ H$



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”

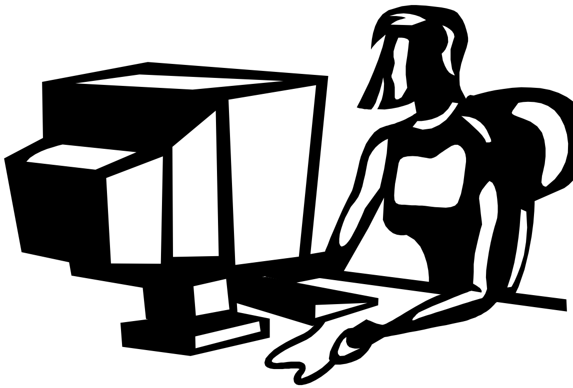
Proof assistant performance (fixes)

- How?
  - More systematically, have good abstraction barriers

Proof assistant performance (fixes)

- How?
  - Have good abstraction barriers

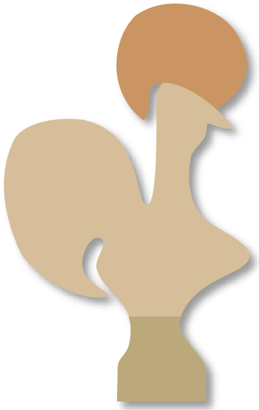
Leaky abstraction barriers generally only torture programmers



Proof assistant performance (fixes)

- How?
  - Have good abstraction barriers

Leaky abstraction barriers torture Coq, too!



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

Proof assistant performance (fixes)

- How?
  - Have good abstraction barriers

Example: Pairing

Two ways to make use of elements of a pair:

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

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

Proof assistant performance (fixes)

- How?
  - Have good abstraction barriers

Example: Pairing

Two ways to make use of elements of a pair:

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

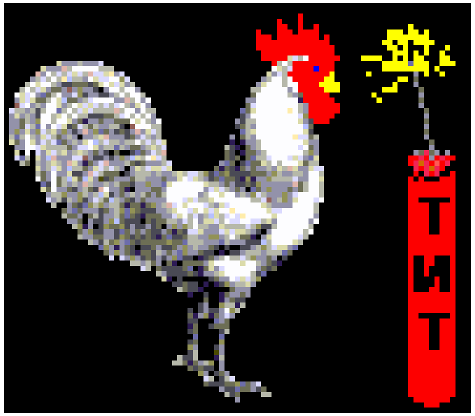
$f\ (\mathrm{let}\ (x, y) := p\ \mathrm{in}\ x)\ (\mathrm{let}\ (x, y) := p\ \mathrm{in}\ y)$. (projections)

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) _ ($\eta_{Y_1} \circ$ f))) $_1$) (only parsing).

Lemma composition_of x y z g f : mor_of _ _ ($f \circ g$) = mor_of y z f $\circ$ mor_of x y g.

Proof.

simpl.

match goal with | [$\vdash$ ((@center ?A?H) $_2$) $_1$ = _] $\Rightarrow$ erewrite (@contr A H (center _; (_; _))) end. 8 s

simpl; reflexivity. 2 s

Grab Existential Variables.

simpl in $*$.

repeat match goal with | [$\vdash$ appcontext[(?x $_2$) $_1$]] $\Rightarrow$ generalize (x $_2$); intro end. 2.5 s

rewrite ?composition_of. 0.5 s

repeat try_associativity_quick (idtac; match goal with | [$\vdash$ appcontext[?x $_1$]] $\Rightarrow$ 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 $\circ$ 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

1 s

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)

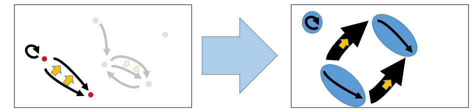
Concrete Example 2 (Generalization)

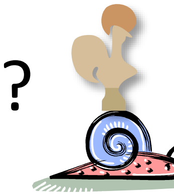
Lemma pseudofunctor_to_cat_assoc_helper $\{x\ x_0 : C\}$ $\{x_2 :$ morphism C x x0$\}$ $\{$x1 $: C\}$
$\{x_5 :$ morphism $C\ x_0\ x_1\}$ $\{x_4 : C\}$ $\{x_7 :$ morphism $C\ x_1\ x_4\}$
$\{p\ p_0 :$ PreCategory$\}$ $\{f :$ morphism $C\ x\ x_4 \rightarrow$ Functor $p_0\ p\}$
$\{p_1\ p_2 :$ PreCategory$\}$ $\{f_0 :$ Functor $p_2\ p\}$ $\{f_1 :$ Functor $p_1\ p_2\}$ $\{f_2 :$ Functor $p_0\ p_2\}$ $\{f_3 :$ Functor $p_0\ p_1\}$ $\{f_4 :$ Functor $p_1\ p\}$
$\{x_{16} :$ morphism $(\_ \rightarrow \_)\ (f\ (x_7 \circ x_5 \circ x_2))\ (f_4 \circ f_3)$%functor$\}$
$\{x_{15} :$ morphism $(\_ \rightarrow \_)\ f_2\ (f_1 \circ f_3)$%functor$\}$ $\{H_2 :$ IsIsomorphism $x_{15}\}$
$\{x_{11} :$ morphism $(\_ \rightarrow \_)\ (f\ (x_7 \circ (x_5 \circ x_2)))\ (f_0 \circ f_2)$%functor$\}$
$\{H_1 :$ IsIsomorphism $x_{11}\}$ $\{x_9 :$ morphism $(\_ \rightarrow \_)\ f_4\ (f_0 \circ f_1)$%functor$\}$ $\{$fst_hyp $: x_7 \circ x_5 \circ x_2 = x_7 \circ (x_5 \circ x_2)\}$
(rew_hyp $: \forall\ x_3 : p_0,$
(idtoiso $(p_0 \rightarrow p)$ (ap f fst_hyp) : morphism_ _ _) $x_3 = x_{11}{}^{-1}\ x_3 \circ (f_0\ {}_1\ (x_{15}{}^{-1}\ x_3) \circ (\mathbb{I} \circ (x_9\ (f_3\ x_3) \circ x_{16}\ x_3))))$
$\{H_0' :$ IsIsomorphism $x_{16}\}$ $\{H_1' :$ IsIsomorphism $x_9\}$ $\{x_{13} : p\}$ $\{x_3 : p_0\}$ $\{x_6 : p_1\}$ $\{x_{10} : p_2\}$
$\{x_{14} :$ morphism $p\ (f_0\ x_{10})\ x_{13}\}$ $\{x_{12} :$ morphism $p_2\ (f_1\ x_6)\ x_{10}\}$ $\{x_8 :$ morphism $p_1\ (f_3\ x_3)\ x_6\}$
: existT $(\lambda\ f_5 :$ morphism $C\ x\ x_4 \Rightarrow$ 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: 100x for the file, from 4m 53s to 28 s
Time spent: a few hours

Outline

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





- **For users (workarounds)**
  - **Arguments vs. fields and packed records**
  - **Proof by duality as proof by unification**
  - **Abstraction barriers**
  - **Proof by reflection**

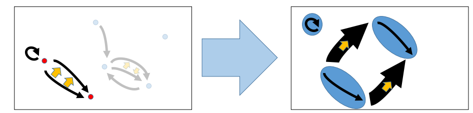
- For developers (features)
  - Primitive Projections
  - Higher inductive types
  - Universe Polymorphism
  - More judgmental rules
  - Hashconsing

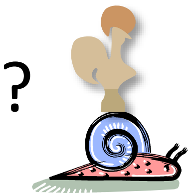
Proof assistant performance (fixes)

Better Proof Assistants

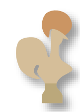
Outline

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

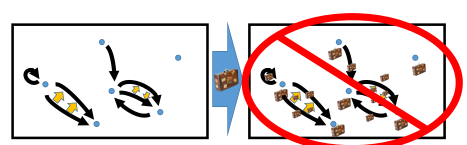


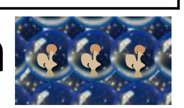


- For users (workarounds)
  - Arguments vs. fields and packed records
  - Proof by duality as proof by unification
  - Abstraction barriers
  - Proof by reflection



- **For developers (features)**
  - **Primitive projections**
  - **Higher inductive types**
  - **Universe Polymorphism**
  - **More judgmental rules**
  - **Hashconsing**





Universes image from Abell NGC2218 hst big, NASA, http://en.wikipedia.org/wiki/Abell_2218#mediaviewer/File:Abell_NGC2218_hst_big.jpg, released in Public Domain; Bubble from http://pixabay.com/en/blue-bubble-shiny-157652/, released in Public Domain CC0, combined in Photoshop by Jason Gross

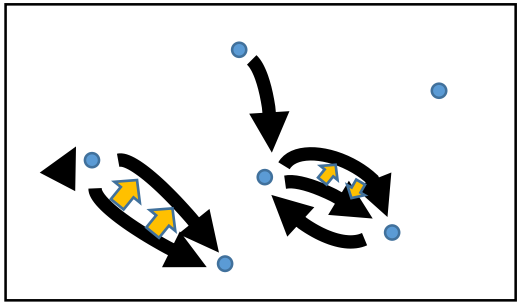
Dam image from http://www.flickr.com/photos/gammaman/7803829282/ by Eli Christman, CC by 2.0
Fence image from http://www.picgifs.com/clip-art/playing-children/clip-art-playing-children-362018-689955/

Proof assistant performance (fixes)

- How?
  - Primitive projections

Proof assistant performance (fixes)

- How?
  - Primitive projections



Definition 2-Graph :=

  { V : Type &

  { 1E : V → V → Type &

    $\forall\, v_1\ v_2,\ 1\mathrm{E}\ v_1\ v_2 \to 1\mathrm{E}\ v_1\ v_2 \to \mathrm{Type}$ }}.

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 :=

$\{\, V \quad : \text{Type} \;\&$

$\{\, 1E \quad : V \to V \to \text{Type} \;\&$

$\forall\, v_1\; v_2,\; 1E\; v_1\; v_2 \to 1E\; v_1\; v_2 \to \text{Type} \,\}.$

Definition V $(G : \text{2-Graph}) :=$ $pr_1\; G$.

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) :=
 @pr₁
   (@pr₁ Type (λ V : Type ⇒
                  { 1E : V → V → Type &
                       ∀ v₁ v₂, 1E v₁ v₂ → 1E v₁ v₂ → Type })
         G →
    @pr₁ Type (λ V : Type ⇒
                  { 1E : V → V → Type &
                       ∀ v₁ v₂, 1E v₁ v₂ → 1E v₁ v₂ → Type })
         G →
    Type)
   (λ 1E : @pr₁ Type (λ V : Type ⇒
                         { 1E : V → V → Type &
                              ∀ v₁ v₂, 1E v₁ v₂ → 1E v₁ v₂ → Type })
                G →
           @pr₁ Type (λ V : Type ⇒
                         { 1E : V → V → Type &
                              ∀ v₁ v₂, 1E v₁ v₂ → 1E v₁ v₂ → Type })
                G →
           Type ⇒
      ∀ v₁ v₂, 1E v₁ v₂ → 1E v₁ v₂ → Type)
   (@pr₂ Type (λ V : Type ⇒
                  { 1E   : V → V → Type &
                       ∀ v₁ v₂, 1E v₁ v₂ → 1E v₁ v₂ → Type }
         G)
```

Proof assistant performance (fixes)

Definition 1E (G : 2-Graph) :=
@pr_1
 (@pr_1 Type (λ V : Type $\Rightarrow$ { 1E : V $\to$ V $\to$ Type & $\forall$ (v_1 : V) (v_2 : V), 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type }) G $\to$
 @pr_1 Type (λ V : Type $\Rightarrow$ { 1E : V $\to$ V $\to$ Type & $\forall$ (v_1 : V) (v_2 : V), 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type }) G $\to$
 Type)
 (λ 1E : @pr_1 Type (λ V : Type $\Rightarrow$ { 1E : V $\to$ V $\to$ Type & $\forall$ (v_1 : V) (v_2 : V), 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type }) G $\to$
 @pr_1 Type (λ V : Type $\Rightarrow$ { 1E : V $\to$ V $\to$ Type & $\forall$ (v_1 : V) (v_2 : V), 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type }) G $\to$
 Type $\Rightarrow$
 $\forall$(v_1 : @pr_1 Type (λ V : Type $\Rightarrow$ { 1E : V $\to$ V $\to$ Type & $\forall$ (v_1 : V) (v_2 : V), 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type }) G)
 (v_2 : @pr_1 Type (λ V : Type $\Rightarrow$ { 1E : V $\to$ V $\to$ Type & $\forall$ (v_1 : V) (v_2 : V), 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type }) G),
 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type)
 (@pr_2 Type (λ V : Type $\Rightarrow$ { 1E : V $\to$ V $\to$ Type & $\forall$ (v_1 : V) (v_2 : V), 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type }) G)
:@pr_1 Type (λ V : Type $\Rightarrow$ { 1E : V $\to$ V $\to$ Type & $\forall$ (v_1 : V) (v_2 : V), 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type }) G $\to$
 @pr_1 Type (λ V : Type $\Rightarrow$ { 1E : V $\to$ V $\to$ Type & $\forall$ (v_1 : V) (v_2 : V), 1E v_1 v_2 $\to$ 1E v_1 v_2 $\to$ Type }) G $\to$
 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.

Proof assistant performance (fixes)

- How?
  - Don’t use setoids

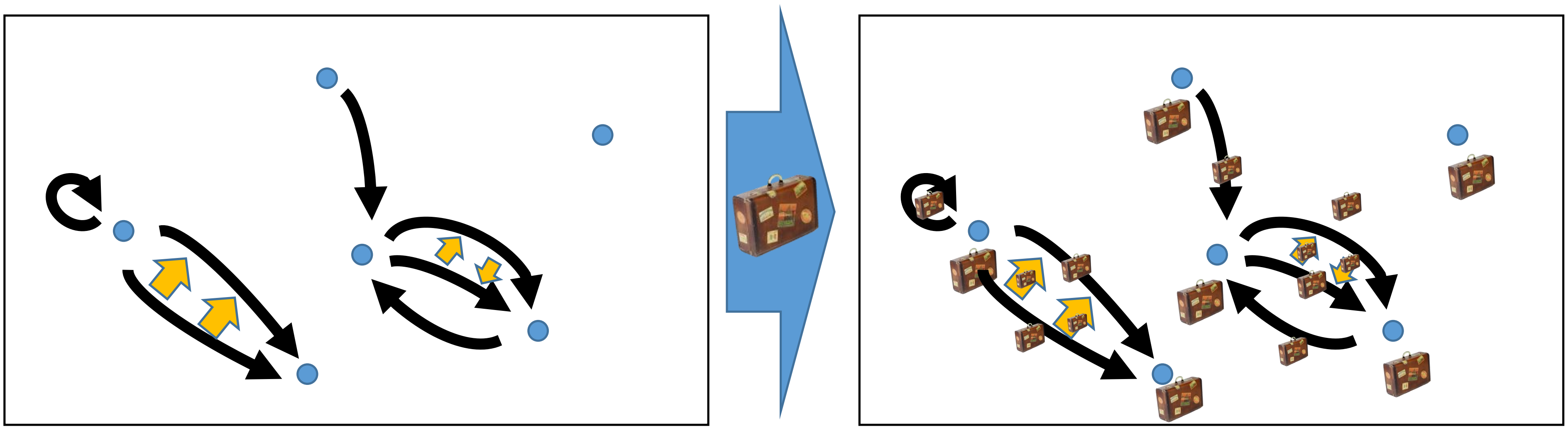
Proof assistant performance (fixes)

- How?
  - Don’t use setoids, use higher inductive types instead!

Proof assistant performance (fixes)

- How?
  - Don't use setoids, use higher inductive types instead!

Setoids add lots of baggage to everything



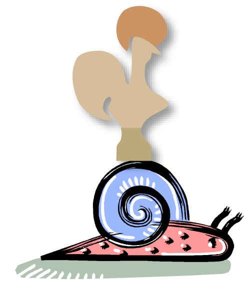
Proof assistant performance (fixes)

- How?
  - Don’t use setoids, use higher inductive types instead!

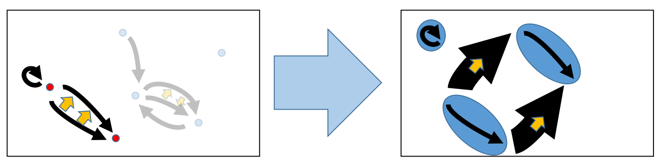
Higher inductive types (when implemented) shove the baggage into the meta-theory, where the type-checker doesn’t have to see it

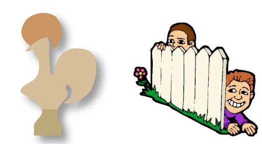
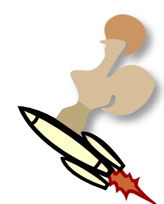
Take-away messages

- 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 paper and presentation will be available at

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

The library is available at

https://github.com/HoTT/HoTT

subdirectory theories/categories

Questions?