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:

Coq, is this correct?

No; here’s a proof of $1 = 0 \rightarrow \text{False}$
How theorem provers should work:

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

Proof: homework
Theorem currying : \((C_1 \to (C_2 \to D)) \cong (C_1 \times C_2 \to D)\).
Proof: trivial.
Qed.
Theorem (currying) : \( (C_1 \to (C_2 \to D)) \cong (C_1 \times C_2 \to D) \)

Proof:
\[
\begin{align*}
\rightarrow: & \quad F \mapsto \lambda (c_1, c_2). F(c_1)(c_2); \text{ morphisms similarly} \\
\leftarrow: & \quad F \mapsto \lambda c_1. \lambda c_2. F(c_1, c_2); \text{ morphisms similarly}
\end{align*}
\]

Functoriality, naturality, and congruence: straightforward.

\(\Box\)

How theorem provers should 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

Functoriality, naturality, and congruence: straightforward.

\[\square\]
Theorem (currying) : \((C_1 \rightarrow (C_2 \rightarrow D)) \cong (C_1 \times C_2 \rightarrow D)\)

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

\(\text{Functoriality, naturality, and congruence: straightforward.}\)

\(\boxed{}\)

How theorem provers do work:

Theorem currying : \((C_1 \rightarrow (C_2 \rightarrow D)) \cong (C_1 \times C_2 \rightarrow D).\)

Proof.

esplit.

\{ by refine \((\lambda_F \ (F \mapsto (\lambda_F \ (c \mapsto F_0 \ c_1 \ c_2)) \ (s \ d \ m \mapsto (F_0 \ d_1)_m \ m_2 \circ (F_m \ m_1)_0 \ s_2)))\)
\((F \ G \ T \mapsto (\lambda_T \ (c \mapsto T \ c_1 \ c_2))))\}. \}

\{ by refine \((\lambda_F \ (F \mapsto (\lambda_F \ (c_1 \mapsto (\lambda_F \ (c_2 \mapsto F_0 \ (c_1, c_2))) \ (s \ d \ m \mapsto F_m \ (1, m)))))\)
\((F \ G \ T \mapsto (\lambda_T \ (c_1 \mapsto (\lambda_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 $\text{Cat} \to \text{Cat}$; and $(C \to D)^{\text{op}} \to (C^{\text{op}} \to D^{\text{op}})$
- the category $\text{Prop}$ of (U-small) hProps
- the category $\text{Set}$ of (U-small) hSets
- the category $\text{Cat}$ of (U-small) strict (pre)categories (strict in the sense of the objects being hSets)
- pseudofunctors
- pseudonatural transformations
- (op)lax comma categories
- profunctors
- identity profunctor (the hom functor $C^{\text{op}} \times C \to \text{Set}$)
- 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
- 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
  - $C \times D \cong D \times C$
  - $C \times 0 \cong 0 \times C \cong 0$
  - $C \times 1 \cong 1 \times C \equiv C$
- Grothendieck construction (oplax colimit) of a pseudofunctor to $\text{Cat}$
- category of sections (gives rise to oplax limit of a pseudofunctor to $\text{Cat}$ when applied to Grothendieck construction)
- functor composition is functorial (there’s a functor $\Delta: (C \to D) \to (D \to)$
- Yoneda lemma
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
- the mathematical content of the library

Cartoon from xkcd, adapted by Alan Huang
Presentation is **not** mainly about:

- category theory or diagram chasing
- the mathematical content of the library
- Coq
Presentation is **not** mainly about:

- category theory or diagram chasing
- 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

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
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)
Durations of Various Tactics vs. Term Size (Coq v8.6, 3.5 GHz Intel i7 CPU, 64 GB RAM)

- 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)
- 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)
- 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:

{ | \text{LCCM}_F := \text{induced}_F (m_{22} \circ m_{12}); \\
\text{LCCM}_T := \lambda_T (\lambda (c : d'_2 / F) \Rightarrow m_{21} c.\beta \circ m_{11} c.\beta) | } =

{ | \text{LCCM}_F := \text{induced}_F m_{12} \circ \text{induced}_F m_{22}; \\
\text{LCCM}_T := \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}) | }
Proof assistant performance (pain)

• When are these slow?
  • When your term is large

• Smallish example (29 000 words): Without Proofs:

{ |
LCCM_F := _\_induced_F (m_{22} \circ m_{12});
LCCM_T := \lambda_T (\lambda (c : d'_2 / F) \Rightarrow m_{21} c. \beta \circ m_{11} c. \beta)
  (\Pi - pf s_2 (\lambda_T (\lambda (c : C) \Rightarrow m_{21} c \circ m_{11} c)
  (\circ_1 - pf m_{21} m_{11})) (m_{22} \circ m_{12})) |
\}

{ |
LCCM_F := _\_induced_F m_{12} \circ _\_induced_F m_{22};
LCCM_T := \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 - pf) (\lambda_T (\lambda (c : d'_2 / F) \Rightarrow m_{21} c. \beta) (\Pi - pf c)
  (\lambda_T (\lambda (c : d'_2 / F) \Rightarrow (d_1)_1 \mathbb{I} \circ m_{11} c. \beta \circ \mathbb{I})
  (\circ_1 - pf) (\lambda_T (\lambda (c : d'_2 / F) \Rightarrow (d_1)_1 \mathbb{I} \circ m_{11} c. \beta \circ \mathbb{I}))
  (\circ_0 - pf) (\lambda_T (\lambda (c : d'_2 / F) \Rightarrow (d_1)_1 \mathbb{I} \circ m_{11} c. \beta \circ \mathbb{I}))
  (\Pi - pf s_2 m_{11} m_{12}) |
\}
Proof assistant performance (pain)

• When are these slow?
  • When your term is large

• Smallish example (29 000 words): Without Proofs:

{\| LCCM_F := \induced F (m_{22} \circ m_{12});
LCCM_T := \lambda_T (\lambda (c : d'_2 / F) \Rightarrow m_{21} c.\beta \circ m_{11} c.\beta)
  (\Pi pf s_2 (\lambda_T (\lambda (c : C) \Rightarrow m_{21} c \circ m_{11} c))
    (\circ_1 pf m_{21} m_{11})) (m_{22} \circ m_{12})) |} =

{\| LCCM_F := \induced_F m_{12} \circ \induced_F m_{22};
LCCM_T := \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 pf (\lambda_T (\lambda (c : d'_2 / F) \Rightarrow m_{21} c.\beta) (\Pi pf d_2 m_{21} m_{22})))
  (\lambda_T (\lambda (c : d'_2 / F) \Rightarrow (d_1)_1 \mathbb{I} \circ m_{11} c.\beta \circ \mathbb{I})
    (\circ_1 pf (\lambda_T (\lambda (c : d'_2 / F) \Rightarrow (d_1)_1 \mathbb{I} \circ m_{11} c.\beta))
      (\circ_0 pf (\lambda_T (\lambda (c : d_2 / F) \Rightarrow m_{11} c.\beta)
        (\Pi pf s_2 m_{11} m_{12})) \mathbb{I}))) \mathbb{I}))) |}
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!
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
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 vertices 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 \to V \to Type \}.

Record IsGraph (V : Type) (E : V \to V \to Type) := \{ \}. 
Proof assistant performance (fixes)

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

Record \text{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}.

\text{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 $\eta$)

Two ways to make use of elements of a pair:

`let (x, y) := p in f x y`. (pattern matching)

`f (fst p) (snd p)`. (projections)
Proof assistant performance (fixes)

• How?
  • Have good abstraction barriers

Example: Pairing (without judgmental $\eta$)

Two ways to make use of elements of a pair:

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

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

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 \( \eta_{Y_1} := \text{IsInitialMorphism.morphism} (@\text{HM} \ Y_1) \) in

(@\text{center} _ (\text{IsInitialMorphism.property} (@\text{HM} \ Y_0) _ (\eta_{Y_1} \circ f))) _ (\text{only parsing}).

Lemma composition_of \( x \ y \ z \ g \ f : \text{mor.of}_{-\ -} (f \circ g) = \text{mor.of}_{y \ z} f \circ \text{mor.of}_{x \ y} g \).

Proof.

simpl.

match goal with | [ ⊢ ((@\text{center} ?A?H) _ 1 = _) ] ⇒ erewrite (@\text{contr} A H (\text{center} _ ; (_ ; _))) end.
simpl; reflexivity.
Grab Existential Variables.
simpl in *.
repeat match goal with | [ ⊢ \text{appcontext}[(?x_2) _ 1 ] ] ⇒ generalize (x_2); intro end.
 rewrite ?composition_of.
repeat try_associativity_quick (idtac; match goal with | [ ⊢ \text{appcontext}[(?x_1) _ ] ] ⇒ simpl rewrite x_2 end).
reflexivity.
Qed.

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 \( \text{mor_of} \ Y_0 \ Y_1 \ f := \)

(let \( \eta_{Y_1} := \text{IsInitialMorphism}_\text{morphism} \ (@HM \ Y_1) \) in

\( \text{IsInitialMorphism}_\text{property}_\text{morphism} \ (@HM \ Y_0) \ _ \ (\eta_{Y_1} \circ f) \) (only parsing).

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

Proof.

simpl.

erewrite \( \text{IsInitialMorphism}_\text{property}_\text{morphism}_\text{unique} \); [ reflexivity | ].

rewrite ?\( \text{composition_of} \).

repeat try_associativity_quick rewrite \( \text{IsInitialMorphism}_\text{property}_\text{morphism}_\text{property} \).

reflexivity.

Qed.

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 o (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_suc in H';

  eapply @trunc_equiv'; [ | exact H' ]); match goal with

  | [ ⊢ appcontext[?m o I] ] ⇒ simpl rewrite (right_identity _ _ _ m)

  | [ ⊢ appcontext[?l o ?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') _1).

  abstract (etransitivity; [ apply ((@center _ H') _2 | auto with morphism ]).

Defined.

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)) := 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' o 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 o (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_property M Y f; IsInitialMorphism_property_morphism_property M Y f)
  m' (@ IsInitialMorphism_property_morphism_unique M Y f m' 1 m' 2) (center _) |}.
Lemma pseudofunctor_to_cat_assoc_helper \{ x \ x_0 : \mathcal{C} \} \{ x_2 : \text{morphism} \ \mathcal{C} \times x_0 \} \{ x_1 : \mathcal{C} \}
\{ x_5 : \text{morphism} \ \mathcal{C} \times x_0 \} \{ x_4 : \mathcal{C} \} \{ x_7 : \text{morphism} \ \mathcal{C} \times x_1 \times x_4 \}
\{ p \ p_0 : \text{PreCategory} \} \{ f : \text{morphism} \ \mathcal{C} \times x_4 \rightarrow \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} \ (\_ \rightarrow \_) \ (f \ (x_7 \circ x_5 \circ x_2)) \ (f_4 \circ f_3) \%	ext{functor} \}
\{ x_{15} : \text{morphism} \ (\_ \rightarrow \_) \ f_2 \ (f_1 \circ f_3) \%	ext{functor} \} \{ H_2 : \text{IsIsomorphism} \ x_{15} \}
\{ x_{11} : \text{morphism} \ (\_ \rightarrow \_) \ (f \ (x_7 \circ (x_5 \circ x_2))) \ (f_0 \circ f_2) \%	ext{functor} \}
\{ H_1 : \text{IsIsomorphism} \ x_{11} \} \{ x_9 : \text{morphism} \ (\_ \rightarrow \_) \ f_4 \ (f_0 \circ f_1) \%	ext{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 , \}
\quad \left( \text{idtoiso} \ (p_0 \rightarrow p) \ (\text{ap} f \ \text{fst\_hyp}) : \text{morphism} \ (\_ \rightarrow \_) \ x_3 = x_{11} \ \circ^{-1} x_3 \circ (f_0 \ \circ (f_{15} \ \circ^{-1} x_3) \circ (\text{Id} \ \circ (x_9 \ (f_3 \ x_3) \circ x_{16} \ x_3))) \right)
\{ 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 \}
\{ \text{existT} (\lambda f_5 : \text{morphism} \ \mathcal{C} \times x_4 \Rightarrow \text{morphism} \ p \ (f \ f_5) \ x_3) \ x_{13} \}
\quad (x_7 \circ x_5 \circ x_2)
\quad (x_{14} \circ (f_0 \ \circ x_{12} \circ x_9 \ x_6) \circ (f_4 \ \circ x_8 \circ x_{16} \ x_3)) = (x_7 \circ (x_5 \circ x_2) ; x_{14} \circ (f_0 \ \circ (f_1 \ \circ x_8 \circ x_{15} \ x_3)) \circ x_{11} \ x_3).

Proof.
\text{helper\_t assoc\_before\_commutes\_tac}.
\text{assoc\_fin\_tac}.
\text{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 :=

\[
\{ V : \text{Type} \& \\
\{ 1E : V \to V \to \text{Type} \& \\
\quad \forall \nu_1 \nu_2, 1E \nu_1 \nu_2 \to 1E \nu_1 \nu_2 \to \text{Type} \} \}.
\]

Definition \( V \) (\( G : 2\)-Graph) := \( \text{pr}_1 G \).

Definition \( 1E \) (\( G : 2\)-Graph) := \( \text{pr}_1 (\text{pr}_2 G) \).

Definition \( 2E \) (\( G : 2\)-Graph) := \( \text{pr}_2 (\text{pr}_2 G) \).
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 (\lambda 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 \to V \to Type \&

\forall v_1, v_2, 1E v_1 v_2 \to 1E v_1 v_2 \to Type \}.

Definition V (G : 2-Graph) := \text{pr}_1 G.

Definition 1E (G : 2-Graph) := \text{pr}_1 (\text{pr}_2 G).
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 }))
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) :=

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

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

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

Recall: Original was:

Definition 1E (G : 2-Graph) := pr1 (pr2 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

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?