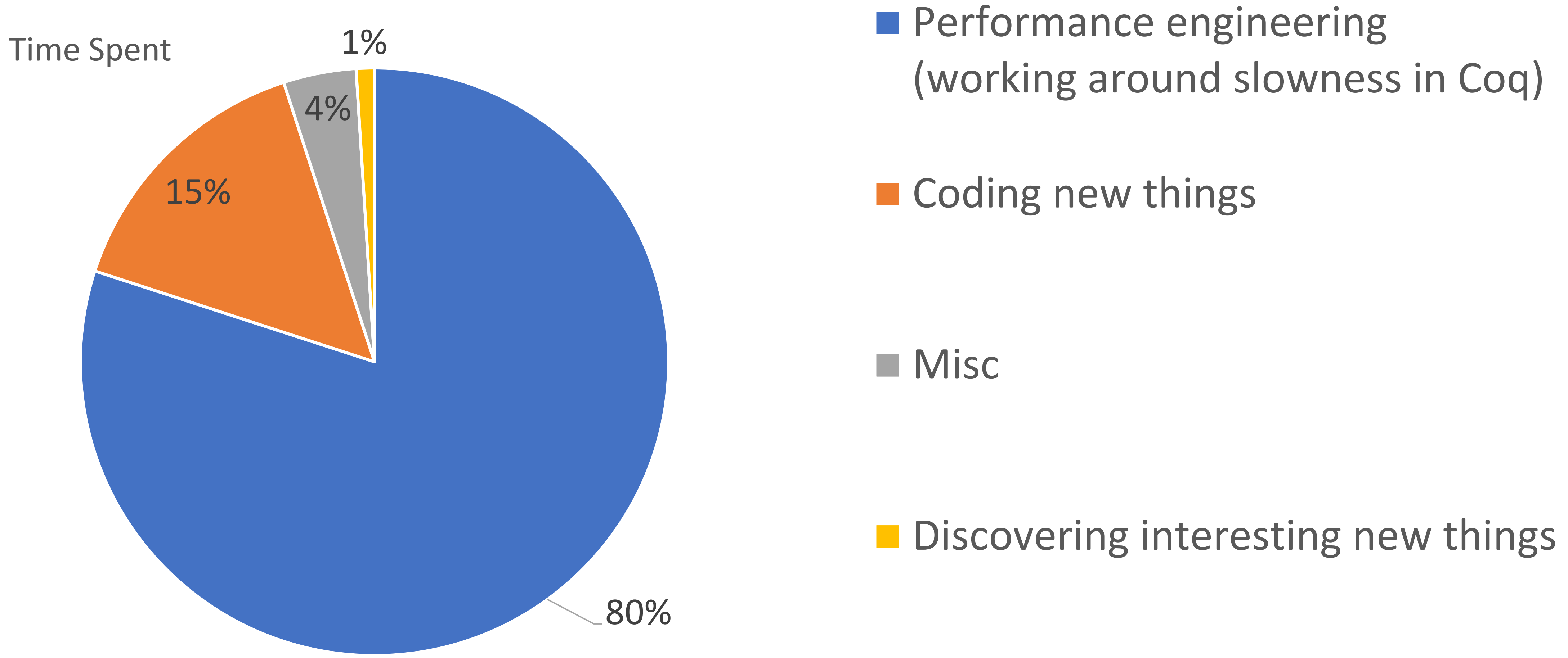
10 Years of Superlinear Slowness in Coq

By **Jason Gross** and Andres Erbsen

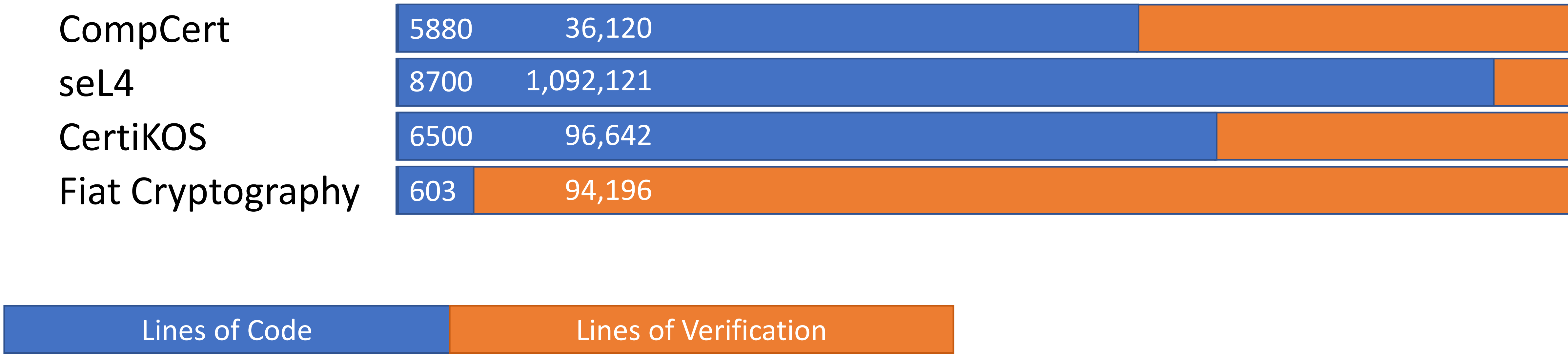
My PhD



Time Spent
1%
4%
15%
80%
Performance engineering (working around slowness in Coq)
Coding new things
Misc
Discovering interesting new things

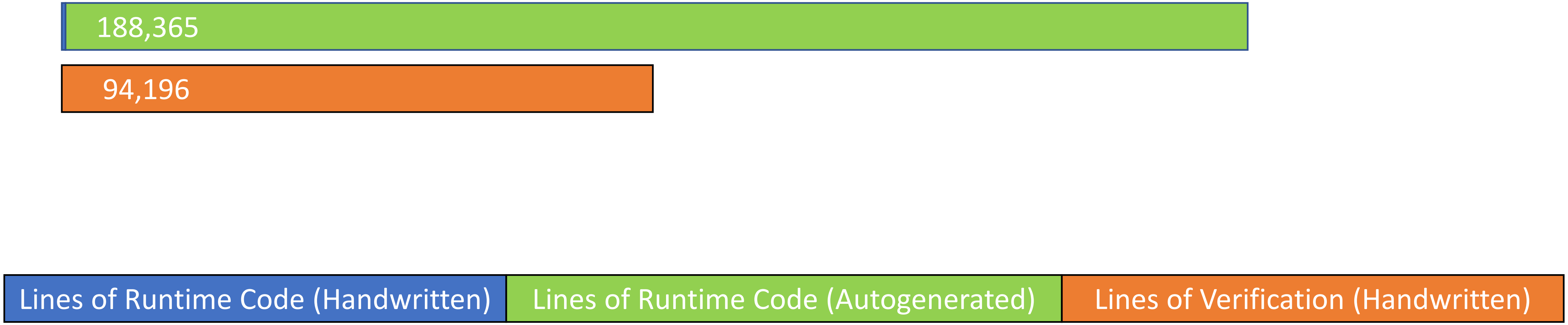
Why We Need Automation: Verification Overhead

- 10x—100x overhead



Automating Verification: ≈ No Marginal Overhead

Fiat Cryptography:



Proof Engine: The API for Automation

Proof engine is:

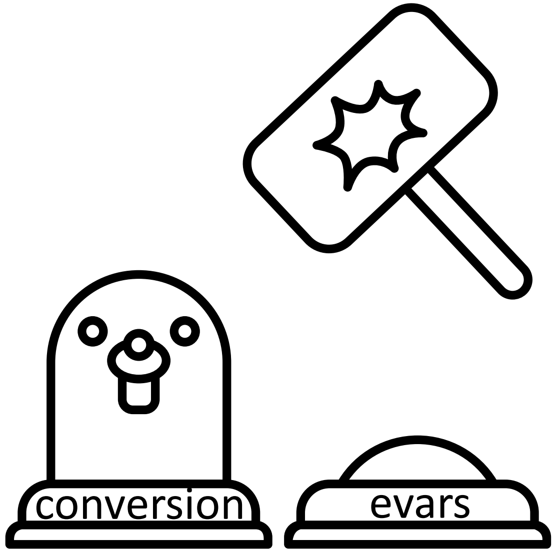
- an interface for programmers: build tactics on top of this
  - e.g., rewrite is not a building block, it's something that should be built
  - would be nice to have minimal, orthogonal, performant interface

Proof assistants prove by typechecking proof terms

- want **modular**, **incremental** construction of proof terms
  - want feedback at the location of error (modularity)
  - want power – everything should be doable in parts (incrementality)
  - want efficiency

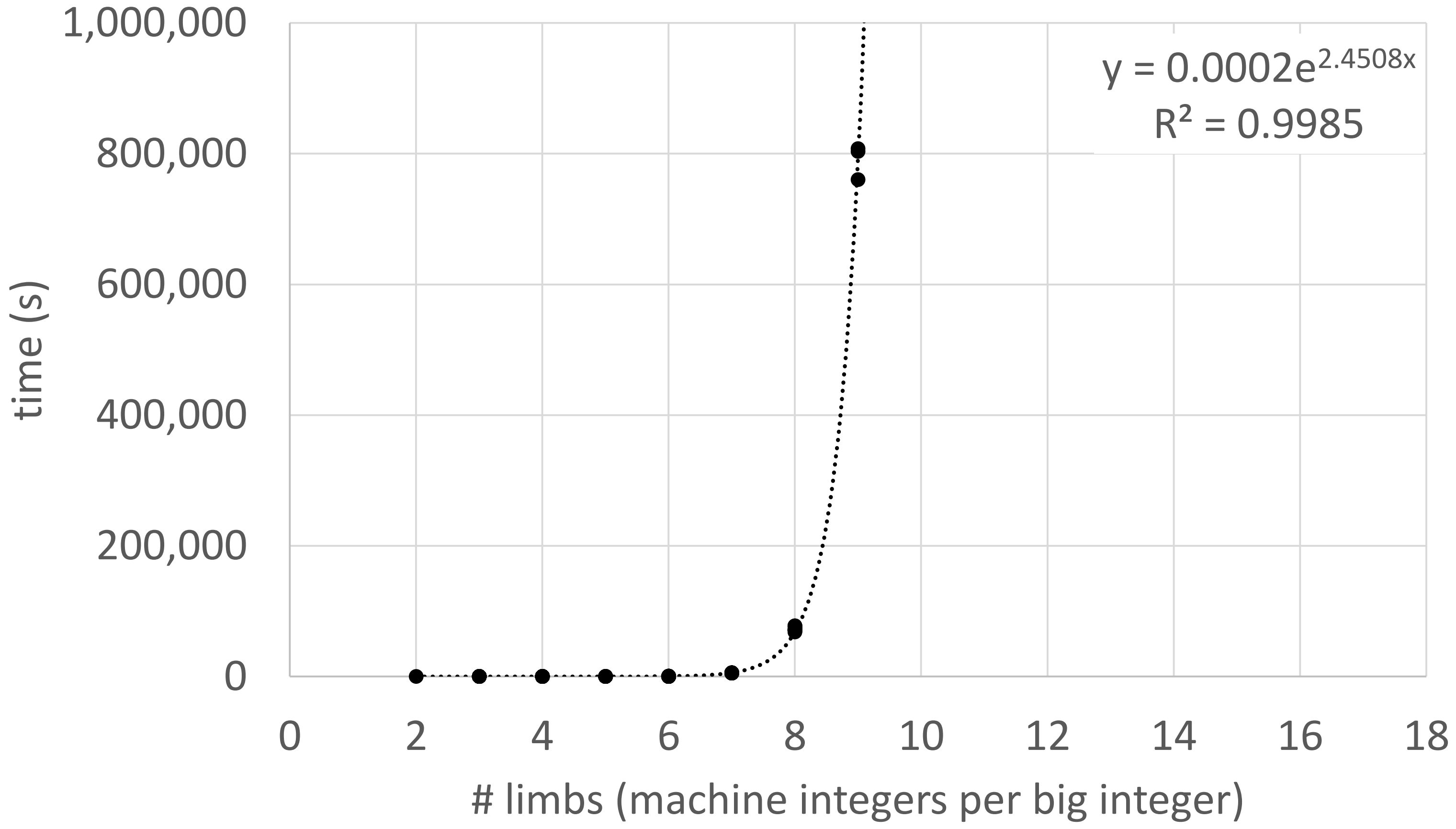
This Presentation

- POPLmark for Proof Engines
- Presentation will sketch the most salient ways we find Coq's current proof engine inadequate
  - Superlinear
  - Whack-a-mole
  - Incoherent



Superlinearity

Superlinearity: 4,000 Millenia is Too Long!



Superlinearity: 4,000 Millenia is Too Long!

What was slow?

`abstract reflexivity`

Why was it slow?

black-box conversion heuristics

What’s Hard About Conversion, By Example

Let’s unify

$$\texttt{fact 10} \equiv 10 \cdot \texttt{fact 9}$$

We have four choices:

1. Heuristically reduce factorial on the left one step (desired in this case)

$$10 \cdot \texttt{fact 9} \equiv 10 \cdot \texttt{fact 9}$$

2. Heuristically reduce multiplication on the right one step (very bad)

$$\texttt{fact 10} \equiv \texttt{fact 9} + 9 \cdot \texttt{fact 9}$$

$$\texttt{fact 10} \equiv \texttt{fact 9} + \texttt{fact 9} + 8 \cdot \texttt{fact 9}$$

…

What’s Hard About Conversion, By Example

Let’s unify

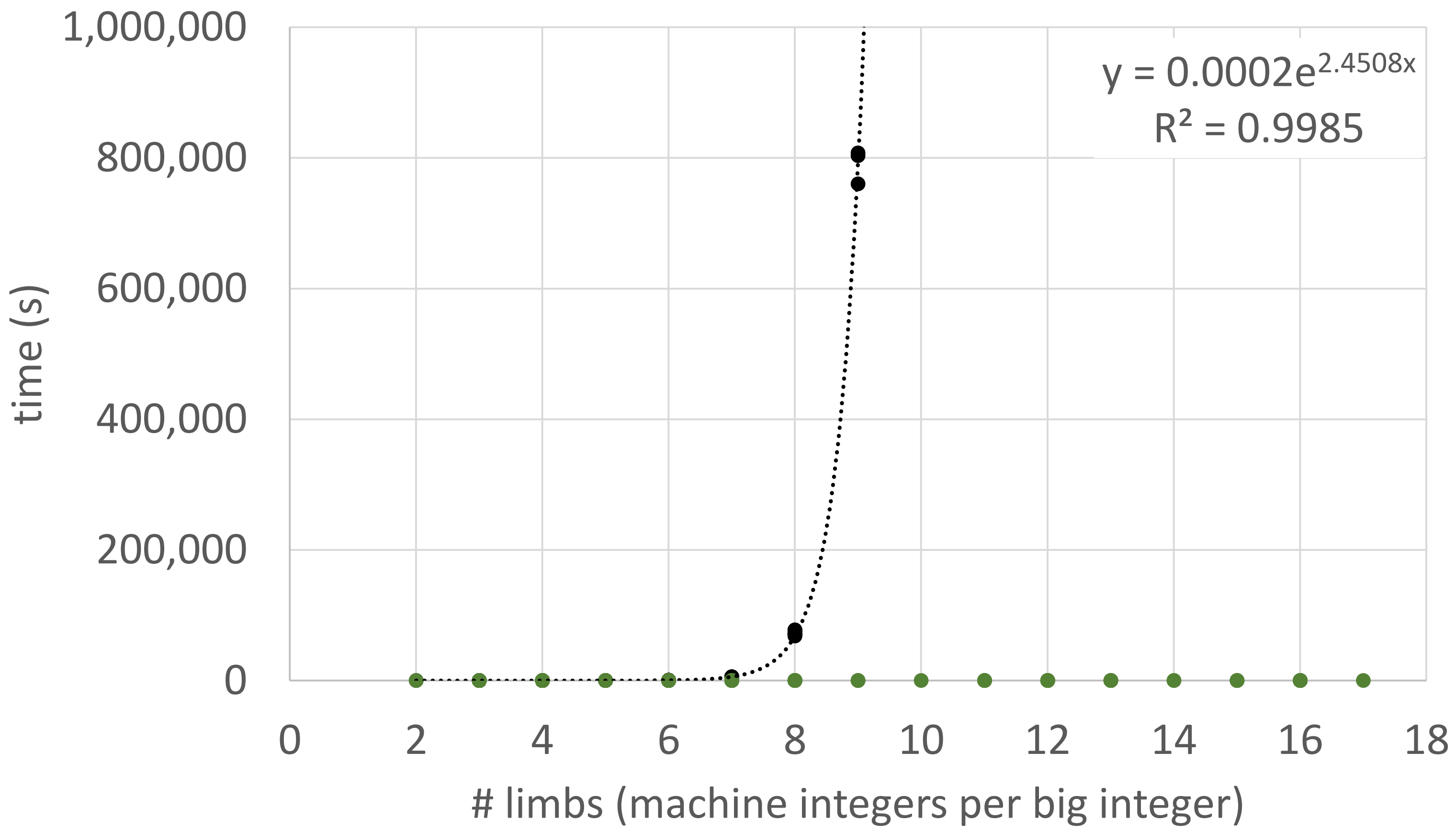
$$\texttt{fact 10} \quad \equiv \quad \texttt{10} \cdot \texttt{fact 9}$$

We have four choices:

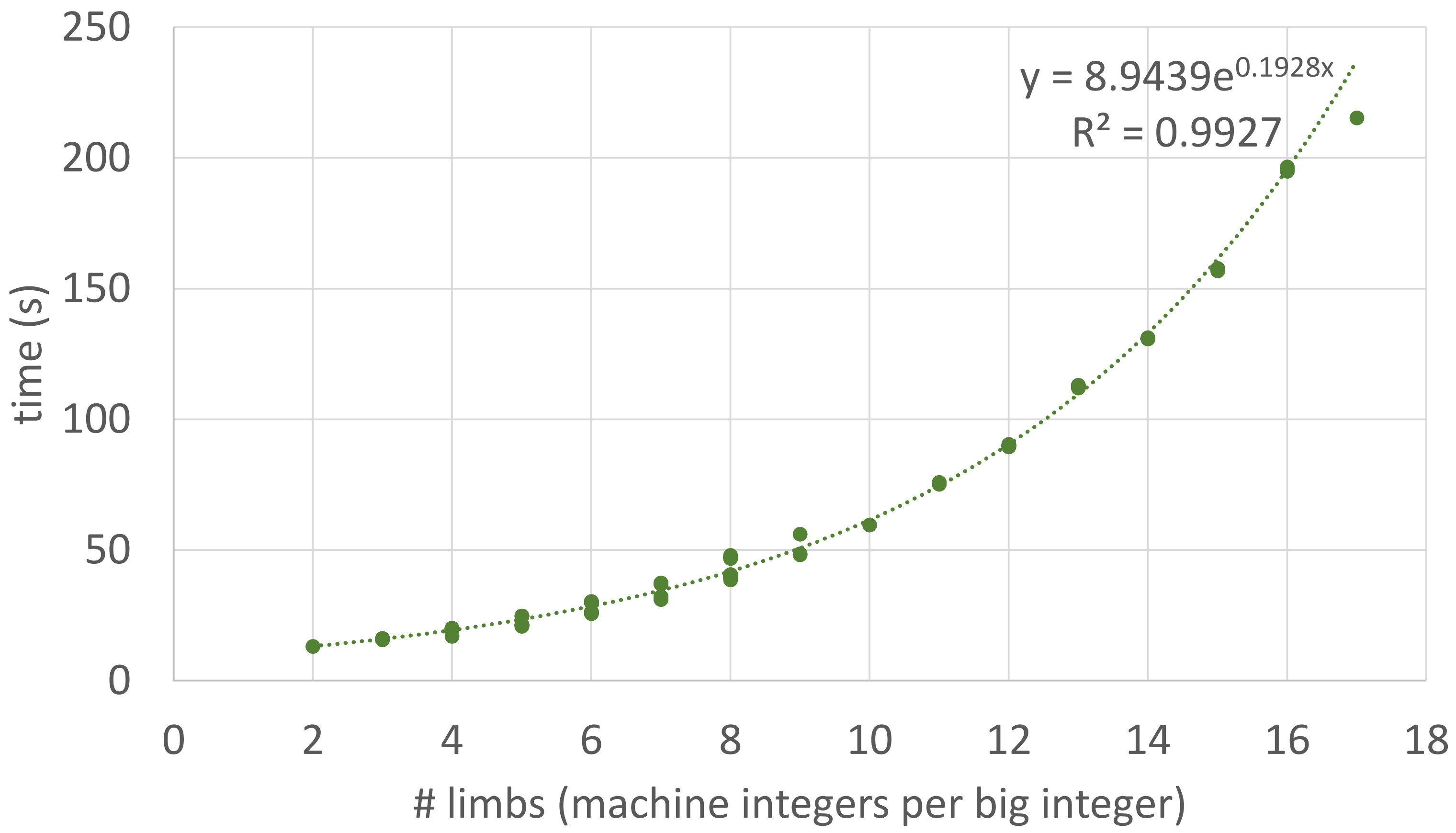
1. Heuristically reduce factorial on the left one step (desired in this case)
2. Heuristically reduce multiplication on the right one step (very bad)
3. Fully reduce both sides (also very slow)
4. Demand conversion hints from the user

Unclear what’s best, but (4) should at least be an option.

Superlinearity: ~~4,000 Millenia~~ is Too Long!



Superlinearity: Exponential is Too Slow ☹



Conversion is Everywhere

- Fixpoint refolding: `cbn`, `simpl`, `hnf`, `intro`, inductive type checking, formerly type inference; performance depends on size of fixpoint body (coq/coq#11887)
- `Extraction` (coq/coq#16172)
- `decide equality` (coq/coq#16290)
- `set` (occurs modulo unification when there are holes)
- `destruct` (inductive type extraction, pattern, convoying, retypechecking goal/context; see also coq/coq#374)
- `clearbody` (needed to ensure well-typing)
- `rewrite` (sometimes desirable, but no way to turn off modulo unification/conversion)
- `apply` (often desirable, but hard to control)

…

Whack-a-Mole: Contexts, Term Size, and Evars

Setup: Imperative code inside weakest precondition (wp) predicate

```
Theorem my_function_correct : forall state,
 my_precondition state
 → weakest_precondition state my_function postcondition.
```

Whack-a-Mole: Contexts, Term Size, and Evars

Real example: verifying chacha20

```
my_function := …; i = 0; while (i < 10) { i += 1;
    x0  += x4 ; x12 ^= x0 ; x12 <<<= 16;
    x8  += x12; x4  ^= x8 ; x4  <<<= 12;
    x0  += x4 ; x12 ^= x0 ; x12 <<<= 8;
    x8  += x12; x4  ^= x8 ; x4  <<<= 7;

    x1  += x5 ; x13 ^= x1 ; x13 <<<= 16;
    x9  += x13; x5  ^= x9 ; x5  <<<= 12;
    x1  += x5 ; x13 ^= x1 ; x13 <<<= 8;
    x9  += x13; x5  ^= x9 ; x5  <<<= 7;

    x2  += x6 ; x14 ^= x2 ; x14 <<<= 16;
    x10 += x14; x6  ^= x10; x6  <<<= 12;
    x2  += x6 ; x14 ^= x2 ; x14 <<<= 8;
    x10 += x14; x6  ^= x10; x6  <<<= 7;

    x3  += x7 ; x15 ^= x3 ; x15 <<<= 16;
    x11 += x15; x7  ^= x11; x7  <<<= 12;
    x3  += x7 ; x15 ^= x3 ; x15 <<<= 8;
    x11 += x15; x7  ^= x11; x7  <<<= 7;

    x0  += x5 ; x15 ^= x0 ; x15 <<<= 16;
    x10 += x15; x5  ^= x10; x5  <<<= 12;
    x0  += x5 ; x15 ^= x0 ; x15 <<<= 8;
    x10 += x15; x5  ^= x10; x5  <<<= 7;

    x1  += x6 ; x12 ^= x1 ; x12 <<<= 16;
    x11 += x12; x6  ^= x11; x6  <<<= 12;
    x1  += x6 ; x12 ^= x1 ; x12 <<<= 8;
    x11 += x12; x6  ^= x11; x6  <<<= 7;

    x2  += x7 ; x13 ^= x2 ; x13 <<<= 16;
    x8  += x13; x7  ^= x8 ; x7  <<<= 12;
    x2  += x7 ; x13 ^= x2 ; x13 <<<= 8;
    x8  += x13; x7  ^= x8 ; x7  <<<= 7;

    x3  += x4 ; x14 ^= x3 ; x14 <<<= 16;
    x9  += x14; x4  ^= x9 ; x4  <<<= 12;
    x3  += x4 ; x14 ^= x3 ; x14 <<<= 8;
    x9  += x14; x4  ^= x9 ; x4  <<<= 7;
} …
```

Whack-a-Mole: Contexts, Term Size, and Evars

…

st_0 := …

```
==========================================
```

$$\text{wp}\left(\text{st}_0, \begin{array}{l}\texttt{x0 += x4 ; x12 ^= x0 ; x12 <<<= 16;} \\ \texttt{x8 += x12; x4 ^= x8 ; x4 <<<= 12;} \\ \texttt{x0 += x4 ; x12 ^= x0 ; x12 <<<= 8 ;} \\ \texttt{x0 += x4 ; x12 ^= x0 ; x12 <<<= 8 ;} \\ \texttt{x8 += x12; x4 ^= x8 ; x4 <<<= 7 ;} \\ \ldots\end{array}, \text{post}\right)$$

Whack-a-Mole: Contexts, Term Size, and Evars

```
…
st₀  := …
v₀   := ?v
st₁  := st₀[x0:=v₀]

==========================================
```

$$\mathrm{wp}\left(\mathtt{st_1}, \begin{array}{l}\mathtt{x0\ \ += x4\ ;\ x12\ \hat{}= x0\ ;\ x12\ <<<= 16;}\\ \mathtt{x8\ \ += x12;\ x4\ \ \hat{}= x8\ ;\ x4\ \ <<<= 12;}\\ \mathtt{x0\ \ += x4\ ;\ x12\ \hat{}= x0\ ;\ x12\ <<<= 8\ ;}\\ \mathtt{x0\ \ += x4\ ;\ x12\ \hat{}= x0\ ;\ x12\ <<<= 8\ ;}\\ \mathtt{x8\ \ += x12;\ x4\ \ \hat{}= x8\ ;\ x4\ \ <<<= 7\ ;}\\ \ldots\end{array}, \mathrm{post}\right)$$

```
exists v0 v4,
     ⟦x0⟧st₀   = Some v0
  /\ ⟦x4⟧st₀   = Some v4
  /\ v0 + v4  =      ?v
```

Whack-a-Mole: Contexts, Term Size, and Evars

```
…
st₀ := …
v₀  := ?v
st₁ := st₀[x0:=v₀]
```

Solve with

```
repeat eexists;
repeat split;
reflexivity
```

```
==========================================================

wp( st₁,           x12 ^= x0 ; x12 <<<= 16;
       x8  += x12; x4  ^= x8 ; x4  <<<= 12;
       x0  += x4 ; x12 ^= x0 ; x12 <<<= 8 ;   , post )
       x0  += x4 ; x12 ^= x0 ; x12 <<<= 8 ;
       x8  += x12; x4  ^= x8 ; x4  <<<= 7 ;
                    ...
```

```
exists v0 v4,
     ⟦x0⟧st₀   = Some v0
  /\ ⟦x4⟧st₀   = Some v4
  /\ v0 + v4  =      ?v
```

Whack-a-Mole: Contexts, Term Size, and Evars

```
…
st₀ := …
v₀  := x + y
st₁ := st₀[x0:=v₀]

==========================================
```

$$\mathtt{wp}\left(\mathtt{st}_1,\ \begin{array}{l} \texttt{x12 ^= x0 ; x12 <<<= 16;} \\ \texttt{x8 += x12; x4 ^= x8 ; x4 <<<= 12;} \\ \texttt{x0 += x4 ; x12 ^= x0 ; x12 <<<= 8 ;} \\ \texttt{x0 += x4 ; x12 ^= x0 ; x12 <<<= 8 ;} \\ \texttt{x8 += x12; x4 ^= x8 ; x4 <<<= 7 ;} \\ \dots \end{array},\ \mathrm{post}\right)$$

Whack-a-Mole: Contexts, Term Size, and Evars

```
…
st₀  := …
v₀   := x + y
st₁  := st₀[x0:=v₀]
v₁   := ?v
st₂  := st₁[x12:=v₁]
========================================
```

```
       (                 x12 ^= x0 ; x12 <<<= 16;       )
       (     x8  += x12; x4  ^= x8 ; x4  <<<= 12;       )
wp     ( st₁, x0  += x4 ; x12 ^= x0 ; x12 <<<= 8 ; , post )
       (     x0  += x4 ; x12 ^= x0 ; x12 <<<= 8 ;       )
       (     x8  += x12; x4  ^= x8 ; x4  <<<= 7 ;       )
       (                       ...                      )
```

```
exists v12 v0,
     ⟦x12⟧_st₀   = Some v12
  /\ ⟦x0 ⟧_st₀   = Some v0
  /\ v12 ^ v0  =      ?v
```

Whack-a-Mole: Contexts, Term Size, and Evars

```
…
st₀ := …
v₀  := x + y
st₁ := st₀[x0:=v₀]
v₁  := ?v
st₂ := st₁[x12:=v₁]
=========================================
```

```
                                   x12 <<<= 16;
       x8  += x12; x4  ^= x8 ; x4  <<<= 12;
wp( st₁, x0  += x4 ; x12 ^= x0 ; x12 <<<= 8 ; , post )
       x0  += x4 ; x12 ^= x0 ; x12 <<<= 8 ;
       x8  += x12; x4  ^= x8 ; x4  <<<= 7 ;
                            ...
```

Solve with
`repeat eexists;`
`repeat split;`
`reflexivity`

```
exists v12 v0,
     ⟦x12⟧st₀   = Some v12
  /\ ⟦x0 ⟧st₀   = Some v0
  /\ v12 ^ v0  =      ?v
```

Whack-a-Mole: Contexts, Term Size, and Evars

```
…
st₀ := …
v₀  := x + y
st₁ := st₀[x0:=v₀]
v₁  := z ^ v₀
st₂ := st₁[x12:=v₁]
=======================================
```

$$
\mathrm{wp}\left(\mathtt{st_1}, \begin{array}{l} \qquad\qquad\qquad\qquad\quad \mathtt{x12\ \lll\!=\ 16;} \\ \mathtt{x8\ \ +=\ x12;\ x4\ \ \hat{}=\ x8\ ;\ x4\ \ \lll\!=\ 12;} \\ \mathtt{x0\ \ +=\ x4\ ;\ x12\ \hat{}=\ x0\ ;\ x12\ \lll\!=\ 8\ ;} \\ \mathtt{x0\ \ +=\ x4\ ;\ x12\ \hat{}=\ x0\ ;\ x12\ \lll\!=\ 8\ ;} \\ \mathtt{x8\ \ +=\ x12;\ x4\ \ \hat{}=\ x8\ ;\ x4\ \ \lll\!=\ 7\ ;} \\ \qquad\qquad\qquad \ldots \end{array}, \mathrm{post} \right)
$$

Whack-a-Mole: Contexts, Term Size, and Evars

```
…
st₀ := …
v₀  := x + y
st₁ := st₀[x0:=v₀]
v₁  := z ^ v₀
st₂ := st₁[x12:=v₁]
=========================================
```

• • •

$$\mathrm{wp}\left(\mathtt{st_1},\ \begin{array}{l} \texttt{x12 <<<= 16;} \\ \texttt{x8 += x12; x4 ^= x8 ; x4 <<<= 12;} \\ \texttt{x0 += x4 ; x12 ^= x0 ; x12 <<<= 8 ;} \\ \texttt{x0 += x4 ; x12 ^= x0 ; x12 <<<= 8 ;} \\ \texttt{x8 += x12; x4 ^= x8 ; x4 <<<= 7 ;} \\ \texttt{...} \end{array},\ \mathrm{post}\right)$$

Performance?

andres-erbsen commented on Apr 29, 2019

Member Author

```
The job exceeded the maximum time limit for jobs, and has been terminated.
The job exceeded the maximum time limit for jobs, and has been terminated.
Fatal error: out of memory.
```

LtacProf!

```
total time:     71.126s
 tactic                                    local  total   calls       max
─────────────────────────────────────────┴──────┴──────┴───────┴─────────┘
─straightline --------------------------  65.8%  92.5%   11109    0.862s
─straightline_cleanup ------------------  31.0%  77.5%   27471    0.103s
─clear (hyp_list) ----------------------  45.2%  45.2%  188438    0.023s
─cbn[interp_binop] in * ----------------   8.4%   8.4%    2807    0.016s
─ensure_free ---------------------------   0.0%   3.0%       0    0.034s
─rename_to_different -------------------   0.0%   3.0%     194    0.033s
─assert_succeeds -----------------------   0.0%   2.9%     194    0.032s
─assert_fails --------------------------   0.0%   2.8%     291    0.032s
─tac -----------------------------------   0.0%   2.8%     194    0.032s
─set (H := Set) ------------------------   2.8%   2.8%     194    0.032s
```

Why `clear`?

lia is slower than omega in large contexts #9848

Closed andres-erbsen opened this issue on Mar 27, 2019 · 6 comments

andres-erbsen commented on Mar 27, 2019 • edited Member

Description of the problem

```
Require Import Lia Omega.

Goal True.
  do 1000 pose proof I.
  Time assert (42 = 42) as _ by Lia.lia. (* 10s *)
  Time assert (42 = 42) as _ by omega. (* .01s *)
  exact I.
Abort. (* Qed is also slow here *)

Goal True.
  do 1000 pose I.
  Time assert (42 = 42) as _ by Lia.lia. (* 10s *)
  Time assert (42 = 42) as _ by omega. (* .01s *)
  exact I.
Abort. (* Qed is also slow here *)
```

Dumb Algorithmics Forced By Ltac

```
Ltac straightline_cleanup :=
  repeat match goal with
  | x : Syntax.cmd |- _ => clear x
  | x : BinNums.Z |- _ => clear x
  | x : unit |- _ => clear x
  | x : bool |- _ => clear x
  | x : list _ |- _ => clear x
  | x : nat |- _ => clear x
  …
```

$$\mathcal{O}(n^3)$$

☹

Closed

Improve perf of straightline_cleanup #266

JasonGross wants to merge 1 commit into `master` from `improve-straightline-clea…`

JasonGross commented 10 days ago • edited

Member Author

I guess this is actually a performance regression

▼ Timing Diff

After	File Name	Before	Change	% Change
69m09.35s	Total Time	68m35.72s	+0m33.62s	+0.81%
8m30.04s	compiler/src/compilerExamples/Softmul.vo	8m18.50s	+0m11.54s	+2.31%
1m10.71s	bedrock2/src/bedrock2Examples/LAN9250.vo	0m59.36s	+0m11.35s	+19.12%
4m34.90s	bedrock2/src/bedrock2Examples/lightbulb.vo	4m25.80s	+0m09.09s	+3.42%
12m19.51s	processor/src/processor/KamiRiscvStep.vo	12m25.98s	-0m06.47s	-0.86%
0m14.24s	bedrock2/src/bedrock2Examples/uint128_32.vo	0m07.97s	+0m06.27s	+78.67%
4m10.42s	deps/riscv-coq/src/riscv/Proofs/DecodeByExtension.vo	4m09.16s	+0m01.25s	+0.50%
3m06.38s	compiler/src/compiler/FlattenExpr.vo	3m07.91s	-0m01.53s	-0.81%
1m18.65s	bedrock2/src/bedrock2Examples/insertionsort.vo	1m17.65s	+0m01.00s	+1.28%
0m57.25s	Kami/Ex/SCMMInv.vo	0m59.12s	-0m01.86s	-3.16%
0m48.73s	bedrock2/src/bedrock2Examples/LiveVerif/swap.vo	0m47.29s	+0m01.43s	+3.04%
0m08.67s	bedrock2/src/bedrock2Examples/FE310CompilerDemo.vo	0m07.17s	+0m01.50s	+20.92%

Can we just skip `clear`?

```
total time:     29.464s
 tactic                                   local  total   calls       max
────────────────────────────────────────┴──────┴──────┴───────┴─────────┘
─straightline --------------------------   0.0%  98.9%    1324    0.588s
─refine (uconstr) ----------------------  79.4%  79.4%     516    0.137s
─straightline_side_condition_solver_inli  79.0%  79.0%      97    0.507s
─straightline_refl ---------------------   1.0%  57.6%     419    0.149s
─straightline_set ----------------------   0.1%  17.9%    1194    0.078s
─straightline_split --------------------   0.1%  17.0%     161    0.067s
─letexists_as --------------------------   0.3%  12.5%      97    0.062s
─straightline_cleanup ------------------   0.3%   5.3%    1324    0.007s
─straightline_cleanup_destruct ---------   3.8%   3.8%    1194    0.005s
─unify (constr) (constr) ---------------   3.4%   3.4%     140    0.014s
─split ---------------------------------   3.3%   3.3%     258    0.016s
─change (x = y) ------------------------   2.3%   2.3%      97    0.015s
─unfold1_cmd_goal ----------------------   0.1%   2.1%     193    0.008s
```

What’s up with `refine`?

```
total time:     29.464s
 tactic                                    local  total   calls       max
──────────────────────────────────────────┴──────┴──────┴───────┴─────────┘

─straightline ---------------------------   0.0%  98.9%    1324    0.588s
─refine (uconstr) -----------------------  79.4%  79.4%     516    0.137s
─straightline_side_condition_solver_inli  79.0%  79.0%      97    0.507s
─straightline_refl ----------------------   1.0%  57.6%     419    0.149s
─straightline_set -----------------------   0.1%  17.9%    1194    0.078s
─straightline_split ---------------------   0.1%  17.0%     161    0.067s
─letexists_as ---------------------------   0.3%  12.5%      97    0.062s
─straightline_cleanup -------------------   0.3%   5.3%    1324    0.007s
─straightline_cleanup_destruct ----------   3.8%   3.8%    1194    0.005s
─unify (constr) (constr) ----------------   3.4%   3.4%     140    0.014s
─split ----------------------------------   3.3%   3.3%     258    0.016s
─change (x = y) -------------------------   2.3%   2.3%      97    0.015s
─unfold1_cmd_goal -----------------------   0.1%   2.1%     193    0.008s
```

It’s `reflexivity` again!

Solve with
```
repeat eexists;
repeat split;
reflexivity
```

```
exists v12 v0,
     ⟦x12⟧_{st_o}   = Some v12
  /\ ⟦x0 ⟧_{st_o}   = Some v0
  /\ v12 ^ v0  =      ?v
```

It's ~~`reflexivity`~~ `refine eq_refl`

Solve with
`repeat eexists;`
`repeat split;`
`refine eq_refl`

```
exists v12 v0,
     ⟦x12⟧_sto   = Some v12
  /\ ⟦x0 ⟧_sto   = Some v0
  /\ v12 ^ v0  =      ?v
```

You say toMAYto, I say toMAHto

`Goal: ?e = t` (at type T)

Tactic	**Running Time**
`reflexivity`	`1.594s`
`refine eq_refl`	`5.847s`
`exact eq_refl`	`5.999s`
`refine (@eq_refl T t)`	`1.701s`
`refine (@eq_refl T ?e)`	`1.746s`
`instantiate (1:=t); reflexivity`	`0.423s`
`instantiate (1:=t); refine eq_refl`	`0.491s`
`instantiate (1:=t); refine (@eq_refl T t)`	`0.414s`

Evars and Holes are Everywhere

- Created whenever a term is not given fully
- Easily forgotten:

`Check (id 7).` ⇝ `Check (@id _ 7).`

- Hold a copy of the entire context(!!!)

Created by:

- Literally every tactic that changes the goal EXCEPT `change`, `move`
- `rewrite` for pattern unification
- Passing terms (`constr`, `open_constr`) to tactics

Reducing Context Size …

`Goal: ?e = t` (at type T)

Tactic	**Time**
`reflexivity`	1.594s
`refine eq_refl`	5.847s
`exact eq_refl`	5.999s
`refine (@eq_refl T t)`	1.701s
`refine (@eq_refl T ?e)`	1.746s
`instantiate (1:=t); reflexivity`	0.423s
`instantiate (1:=t); refine eq_refl`	0.491s
`instantiate (1:=t); refine (@eq_refl T t)`	0.414s

Reducing Context Size …

`Goal: ?e = t` (at type T)

Tactic	**Time**	**Less Time**
`reflexivity`	1.594s	0.041s
`refine eq_refl`	5.847s	0.132s
`exact eq_refl`	5.999s	0.131s
`refine (@eq_refl T t)`	1.701s	0.047s
`refine (@eq_refl T ?e)`	1.746s	0.049s
`instantiate (1:=t); reflexivity`	0.423s	0.092s
`instantiate (1:=t); refine eq_refl`	0.491s	0.094s
`instantiate (1:=t); refine (@eq_refl T t)`	0.414s	0.083s

Optimized Proof of Entire Loop Body

total time: 9.577s (vs original 71.126s)

tactic	local	total	calls	max
─straightline ---------------------------	0.1%	90.5%	1324	0.168s
─straightline_side_condition_solver_inli	60.8%	60.8%	97	0.147s
─straightline_refl ---------------------	1.5%	36.4%	419	0.041s
─refine (uconstr) ----------------------	31.9%	31.9%	516	0.037s
─refine_eq_refl_v ----------------------	0.2%	25.4%	0	0.031s
─vm_compute ----------------------------	23.6%	23.6%	161	0.030s
─straightline_set ----------------------	1.0%	22.6%	1194	0.032s
─straightline_split --------------------	0.3%	19.1%	161	0.037s
─unfold1_cmd_goal ----------------------	0.2%	10.4%	193	0.022s
─straightline_cleanup ------------------	1.1%	9.3%	1324	0.019s
─change G ------------------------------	9.2%	9.2%	484	0.022s
─straightline_unfold -------------------	0.8%	8.1%	1097	0.027s

Example 3: Rewriting is Hard

- Dumb reasons
- Interesting technical reasons
- Reasons that contain research-level technical challenges

Example 3: Rewriting is Slow

Dumb reasons (some now fixed, others still open)

- evar use: #6101, #8304, #8822, #12524, #13576, #13708
- uncontrollable unification and conversion: #7933, #9283, #9286, #9287, #10969, #13338, #13346, #15593
- invoking `simpl` refolding when reducing types to `hnf`: #6101, #8304
- typeclasses: #3795, #3921, #15596, #15701, #15747
- universes: #11973, #14488
- undiagnosed: #4977, #8823

Example 3: Rewriting is Hard

Interesting technical reasons

Example: hard to build a rewrite that doesn't duplicate entire goal for every rewrite location

Example 3: Rewriting is Hard

Reasons that contain research-level technical challenges

When binders are involved:

- generate a linearly-sized proof term (tricky)
- in linear time (more tricky)
- using only well-typed incremental modular primitives (open problem)
- with linear time Qed (open problem)

What we hope you now believe

- Performant modular proof engines are important
- We don’t have one yet, and don’t even know how to spec one yet
- POPLmark for proof engines would be a great first step!
- Looking for underlying building blocks, not just a faster rewrite/program logic solver
  - Should work for structured conversion, program logic, etc
  - Building a framework to evaluate these primitives is part of the quest (POPLmark for proof engines)

Discussion & Questions

Our questions:

- What are the right (incremental modular) building blocks for conversion?
- How should binders and contexts be represented?
- What asymptotic performance is “adequate” for the various building blocks?

Your questions: … ?

Extra Content

Rewriting Pseudocode: With Binders

```
rw (fun x:T => e) =
   let rrw := (fun x:T => rw e) in
   let mid := (fun y:T => let (e', _) := beta (rrw y) in e') in
   let f_mid := functional_extensionality
                  (fun z:T => let (_, e'e) := beta (rrw z) in e'e) in
   match rwh mid with
   | (result, mid_result) => (result, eq_trans f_mid mid_result)
   | _ => (mid, f_mid)
   end
```

Rewriting Pseudocode: No Binders

```
rw (f x) =
   let (mid, fx_mid) :=
     match rw f, rw x with
     | (f', f'f), (x', x'x) => (f' x', app_cong f'f x'x)
     | _ => (f x, eq_refl (f x))
     end in
   match rwh mid with
   | (result, mid_result) => (result, eq_trans fx_mid mid_result)
   | _ => (mid, fx_mid)
   end
```

Rewrite Performance With Binders