

the ergonomics of PL metatheory in Lean

7 June 2024

the ergonomics of PL metatheory in Lean

or: I couldn't convince people to use Agda for metatheory
but maybe I can convince you to try out Lean instead

this talk is about:

- tooling and editor support
- language features designed to take advantage of support
- demonstrating this with a real PL metatheory codebase

this talk is not about:

- how to do metatheory in Lean
- Lean's type theory
- (okay maybe a little at the end. but I promise no typing rules)

Coq and Lean are:

- interactive proof assistants
- based on dependent types (proofs are terms)
- with automation via tactics

my issues with Coq

- P1: proof **reading** is missing ephemeral information
- P2: term **writing** isn't incremental
- P3: proof **editing** is sequential
- P4: information for **learning** isn't well integrated

specifically using:

- Coq-LSP
- also applies to VSCoq 1/CoqIDE
- (I don't use emacs so idk Proof General)

P1: proof reading is missing ephemeral information

- Q1: what changes did this tactic make?
 - in `tac1; tac2`, what changes did `tac1` make?
- Q2: what is the type of this subexpression?
 - what is the *context* of this subexpression?
- Q3: what case is this subproof for?

P1: proof reading is missing ephemeral information

- Q1: what changes did this tactic make?
 - in `tac1; tac2`, what changes did `tac1` make?
- Q2: what is the type of this subexpression?
 - what is the *context* of this subexpression?
- Q3: what case is this subproof for?

```
per.v 1 x
theories > per.v > ...
386
387 Lemma InterpUnivN_eta_right i A B R
388   (h : [ tPi A B ] i ∨ R) :
389   forall a b, R a (tAbs (tApp (b ⟨S⟩) (var_tm 0))) ↔ R a b.
390 Proof.
391   move : (InterpUnivN_eta_left _ _ _ _ h) ⇒ ih.
392   split ⇒ *;
393   eapply InterpUnivN_sym; eauto;|
394   apply ih;
395   eapply InterpUnivN_sym; eauto.
396 Qed.
```

```
Goals x
per.v:393:33
Goals (1)
Goal (1)
i : fin
A, B : tm
R : tm_rel
h : [ tPi A B ] i ∨ R
ih : forall a b : tm,
      R (tAbs (tApp a ⟨S⟩ (var_tm 0))) b ↔ R a b

forall a b : tm,
R a (tAbs (tApp b ⟨S⟩ (var_tm 0))) ↔ R a b
```

P1: proof reading is missing ephemeral information

- Q1: what changes did this tactic make?
 - in tac1; tac2, what changes did tac1 make?
- Q2: what is the type of this subexpression?
 - what is the *context* of this subexpression?
- Q3: what case is this subproof for?

The screenshot displays the Lean IDE interface. On the left, the proof script for the lemma `InterpUnivN_eta_right` is shown. The script includes a `forall` statement with a subexpression `(b <S>)` circled in blue. The proof is followed by a `Proof.` block with several tactics: `move`, `split`, `eapply`, `apply`, and `Qed.`

On the right, the Goals panel shows the current goal state at line 389:31, indicating that there are no goals at this point in the proof.

```
per.v 1 x
theories > per.v > InterpUnivN_eta_right
386
387 Lemma InterpUnivN_eta_right i A R D
388 (h : [ tPi A B ] i v R) : ((h 386, c: 0 | o: 12480)--{l: 388, c: 61 | o: 12615 })
389 forall a b, R a (tAbs (tApp (b <S>) (var_tm 0))) ↔ R a b.
390 Proof.
391   move : (InterpUnivN_eta_left _ _ _ h) => ih.
392   split => *;
393   eapply InterpUnivN_sym; eauto;
394   apply ih;
395   eapply InterpUnivN_sym; eauto.
396 Qed.
```

Goals x

▼ per.v:389:31

No goals at this point!

► Messages (0)

P2: term writing isn't incremental

- Q1: how do I fill in this term later?
- Q2: how do I fill in this term with tactics?
 - if I don't know the exact tactic I need?
 - if I need multiple tactics/generate subgoals?

P2: term writing isn't incremental

- Q1: how do I fill in this term later?
- Q2: how do I fill in this term with tactics?
 - if I don't know the exact tactic I need?
 - if I need multiple tactics/generate subgoals?

The screenshot shows a Lean IDE interface. The left pane displays the source code for a theory named 'tstar'. The code defines a function 'tstar' that takes a term 'a' and returns another term. The function is defined using a 'match a with' construct with several cases: 'tUniv', 'tPi', 'tAbs', 'tApp', and a catch-all case '(* ... *)'. The 'tApp' case is highlighted with a blue circle. Below the function definition, there is a lemma 'Par_triangle' and its proof, which uses the 'tstar' function. The right pane shows the 'Goals' section, which is currently empty, indicating that no goals are present at this point in the proof. Below the goals, there is an 'Errors' section with a message: 'Non exhaustive pattern-matching: no clause found for pattern tEq _ _ _;'.

```
509 (* Takahashi translation *)
510 Fixpoint tstar (a : tm) : tm :=
511   match a with
512   | var tm i => a
513   | tUniv   => a
514   | tPi A B => tPi (tstar A) (tstar B)
515   | tAbs a  => tAbs (tstar a)
516   | tApp (tAbs a) b => (tstar a) ..
517   | tApp a b => tApp (tstar a) (tstar b)
518   (* ... *)
519   end.
520
521 Lemma Par_triangle a : forall b, (a => b) -> (b => tstar a).
522 Proof.
523   apply tstar_ind; hauto lq:on inv:Par use:Par_refl,Par_cong,F
524 Qed.
525
```

Goals

▼ join.v:516:36

No goals at this point!

► Messages (0)

Errors:

Non exhaustive pattern-matching: no clause found for pattern
tEq _ _ _;

P2: term writing isn't incremental

- Q1: how do I fill in this term later?
- Q2: how do I fill in this term with tactics?
 - if I don't know the exact tactic I need?
 - if I need multiple tactics/generate subgoals?

The screenshot shows a Lean IDE interface. The top bar displays two files: `per.v 9+, M` and `join.v 7, M`. The main editor shows the following code:

```
509 (* Takahashi translation *)
510 Fixpoint tstar (a : tm) : tm :=
511   match a with
512   | var tm i => a
513   | tUniv   => a
514   | tPi A B => tPi (tstar A) (tstar B)
515   | tAbs a  => tAbs (tstar a)
516   | tApp (tAbs a) b => (tstar a) [..]
517   | tApp a b => tApp (tstar a) (tstar b)
518   | _ => _
519   end
520
521 Lemma Par_triangle a : forall b, (a → b) → (b → tstar a).
522 Proof.
523   apply tstar_ind; eauto lq:on inv:Par use:Par_refl,Par_cong,F
524 Qed.
525
```

The line `| _ => _` at line 518 is circled in blue. The right-hand pane shows the goal state:

```
Goals
▼ join.v:518:11
No goals at this point!
► Messages (0)

Errors:
Unable to satisfy the following constraints:
In environment:
tstar : tm → tm
a0, t, b, a : tm

?x : "tm"
;
```

P2: term writing isn't incremental

- Q1: how do I fill in this term later?
- Q2: how do I fill in this term with tactics?
 - if I don't know the exact tactic I need?
 - if I need multiple tactics/generate subgoals?

The screenshot shows a theorem prover interface with two main panes. The left pane displays code for a function `tstar` with a `refine` block containing a `match` expression. The right pane shows a list of goals, with the first goal being `tstar : tm → tm` and subsequent goals being `tm`.

```
per.v 9+, M | join.v 2, M ×
```

theories > join.v > ...

```
509 (* Takahashi translation *)
510 Fixpoint tstar (a : tm) : tm.
511   refine
512     (match a with
513      | var_tm i ⇒ a
514      | tUniv _ ⇒ a
515      | tPi A B ⇒ tPi (tstar A) (tstar B)
516      | tAbs a ⇒ tAbs (tstar a)
517      | tApp (tAbs a) b ⇒ (tstar a) [_ ..]
518      | tApp a b ⇒ tApp (tstar a) (tstar b)
519      | _ ⇒ _
520     end).
521 Admitted.
522
```

Goals

```
tstar : tm → tm
a0, t, b, a : tm
```

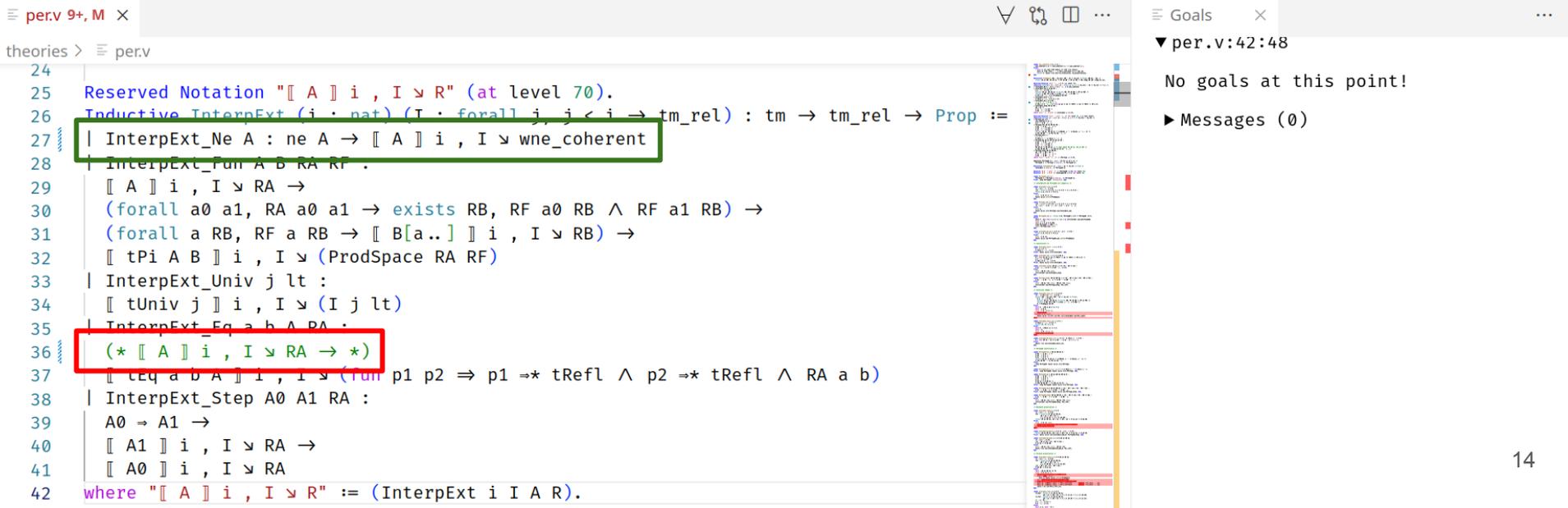
```
tm
▶ Goal (2)
tm
▶ Goal (3)
tm
▶ Goal (4)
tm
▶ Goal (5)
tm
```

P3: proof editing is (more or less) sequential

- Q1: when I change a definition, what proofs are broken?
 - did the proofs break in an expected way? or did I change the wrong definition?
 - how much work is left? what should I repair next?
- Q2: when I change a definition, what proof *cases* are broken?
 - am I missing a new case?
 - is an old case no longer covered by automation?

P3: proof editing is (more or less) sequential

- Q1: when I change a definition, what proofs are broken?
 - did the proofs break in an expected way? or did I change the wrong definition?
 - how much work is left? what should I repair next?



The screenshot shows a proof assistant interface with a code editor on the left and a goals panel on the right. The code editor displays a definition for `InterpExt` with several lines highlighted in green and red. The goals panel shows a single goal with the message "No goals at this point!".

```
per.v 9+, M x
theories > per.v
24
25 Reserved Notation "[[ A ]] i , I ∋ R" (at level 70).
26 Inductive InterpExt (i : nat) (I : forall j, j < i → tm_rel) : tm → tm_rel → Prop :=
27 | InterpExt_Ne A : ne A → [[ A ]] i , I ∋ wne_coherent
28 | InterpExt_Fun A B RA RF :
29   [[ A ]] i , I ∋ RA →
30   (forall a0 a1, RA a0 a1 → exists RB, RF a0 RB ∧ RF a1 RB) →
31   (forall a RB, RF a RB → [[ B[a..] ]] i , I ∋ RB) →
32   [[ tPi A B ]] i , I ∋ (ProdSpace RA RF)
33 | InterpExt_Univ j lt :
34   [[ tUniv j ]] i , I ∋ (I j lt)
35 | InterpExt_Eq a b A RA :
36 (* [[ A ]] i , I ∋ RA → *)
37 [[ tEq a b A ]] i , I ∋ (fun p1 p2 => p1 => tRefl ∧ p2 => tRefl ∧ RA a b)
38 | InterpExt_Step A0 A1 RA :
39   A0 => A1 →
40   [[ A1 ]] i , I ∋ RA →
41   [[ A0 ]] i , I ∋ RA
42 where "[[ A ]] i , I ∋ R" := (InterpExt i I A R).
```

Goals

▼ per.v:42:48

No goals at this point!

► Messages (0)

P3: proof editing is (more or less) sequential

- Q1: when I change a definition, what proofs are broken?
 - did the proofs break in an expected way? or did I change the wrong definition?
 - how much work is left? what should I repair next?

The screenshot shows a proof editor interface with a main workspace on the left and a goal pane on the right. The main workspace contains a lemma and its proof. The lemma is:

```
242 Lemma InterpExt_fwd_R i I A R a0 a1 b0 b1
243 (h : [ A ] i , I  $\triangleright$  R)
244 (hI : forall j lt a0 a1 b0 b1,
245 | a0  $\Rightarrow$  a1  $\rightarrow$  b0  $\Rightarrow$  b1  $\rightarrow$ 
246 | I j lt a0 b0  $\rightarrow$  I j lt a1 b1)
247 (ra : a0  $\Rightarrow$  a1) (rb : b0  $\Rightarrow$  b1) :
248 R a0 b0  $\rightarrow$  R a1 b1.
```

The proof starts with:

```
249 Proof.
250 move : a0 a1 b0 b1 ra rb.
251 elim : A R /h  $\Rightarrow$  //.
252 - move  $\Rightarrow$  ??? ihRB * > *.
253 eapply ihRB; eauto;
254 | last by hauto lq: on unfold: ProdSpace.
255 all: hauto q: on ctrs: Par use: Par_refl.
256 - move  $\Rightarrow$  > > rp rq [rpRefl0] [rqRefl0] RAAb.
257 have [p' [rpRefl1 rp1]] := Pars_confluent _ _ rpRefl0 (rtc_0
258 have [q' [rqRefl1 rq1]] := Pars_confluent _ _ rqRefl0 (rtc_0
259 sauto l: on use: Pars_refl_inv.
260 Qed.
```

The goal pane on the right shows the current goal (1):

```
▼ per.v:251:23
▼ Goals (3)
▼ Goal (1)
i : fin
I : forall j : fin, j < i  $\rightarrow$  tm_rel
hI : forall (j : fin) (lt : j < i) (a0 a1 b0 b1 : tm),
a0  $\Rightarrow$  a1  $\rightarrow$ 
b0  $\Rightarrow$  b1  $\rightarrow$  I j lt a0 b0  $\rightarrow$  I j lt a1 b1
```

Below goal (1), there is a goal (2):

```
► Goal (2)
forall (A B : tm) (RA : tm_rel) (RF : tm  $\rightarrow$  tm_rel  $\rightarrow$ 
Prop),
[ A ] i , I  $\triangleright$  RA  $\rightarrow$ 
(forall a0 a1 b0 b1 : tm,
a0  $\Rightarrow$  a1  $\rightarrow$  b0  $\Rightarrow$  b1  $\rightarrow$  RA a0 b0  $\rightarrow$  RA a1 b1)  $\rightarrow$ 
(forall a0 a1 : tm,
```

P4: information isn't well integrated

- Q1: what does this tactic do?
- Q2: how do I use this syntax?
 - let's see *you* try to remember full syntax for match off the top of your head

claim: Lean's tooling & design
help solve these problems

P4: information isn't well integrated

- Q1: what does this tactic do?
- Q2: how do I use this syntax?

A: documentation on hover!

- bonus: cross-file ctrl-click navigation

P1: proof reading is missing ephemeral information

- Q1: what changes did this tactic make?
 - in `tac1; tac2`, what changes did `tac1` make?
- Q2: what is the type of this subexpression?
 - what is the *context* of this subexpression?

A: infoview at cursor!

- Q3: what case is this subproof for?
A3: *tagged cases* help retain this info

P2: term writing isn't incremental

- Q1: how do I fill in this term later?

A1: with *typed holes*

- Q2: how do I fill in this term with tactics?

A2: by dropping into *proof mode, anywhere*

P3: proof editing is (more or less) sequential

- Q1: when I change a definition, what proofs are broken?
 - did the proofs break in an expected way? or did I change the wrong definition?
 - how much work is left? what should I repair next?
- Q2: when I change a definition, what proof *cases* are broken?
 - am I missing a new case?
 - is an old case no longer covered by automation?

A: [demo]

tactics I enjoy

- `split`: pushes a goal inside of a match
- `calc`: explicit equational reasoning
- `apply_rules [lem, ...]`: keep applying lemmas and hypotheses without backtracking (like Coq's `eauto ... using ...`)

other tools and libraries

for Coq

- SSReflect
- CoqHammer
- Mathcomp
- Psatz

for Lean

- LeanSSR
<https://github.com/verse-lab/lean-ssr>
- Aesop
<https://github.com/leanprover-community/aesop>
- Mathlib
<https://github.com/leanprover-community/mathlib4>
- (n)Linearith (part of Mathlib)

my issues with Lean

basically designed specifically targeting mathematics

- no structural recursion on inductive predicates
 - inductive *propositions* are fine. smth abt decreasing measures
 - pretty sure this is a bug
- no induction tactic on mutual inductives
 - encode mutual inductives as single inductives
 - Coq doesn't have great mutual inductive support anyway
- no coinduction
 - RIP itrees folks ig
- no modules
 - use typeclasses
- no Agda-style unification
 - :(((

ported existing development in ~1 week

TT-model / TT-model /

ionathanch Update README.md b49b0be · 5 days ago History

Name	Last commit message	Last commit da...
..		
README.md	Update README.md	5 days ago
example.lean	[Lean] Generalize levels to any well order with unbounded successors	last week
level.lean	[Lean] The property of always having a larger element is in typeclass...	last week
reduction.lean	[Lean] Use let instead of match where possible	5 days ago
semantics.lean	[Lean] Use let instead of match where possible	5 days ago
soundness.lean	[Lean] Use let instead of match where possible	5 days ago
syntactics.lean	[Lean] Use a bit more simp	5 days ago
typing.lean	[Lean] The property of always having a larger element is in typeclass...	last week

README.md

Mechanization of consistency, Lean edition

This is a Lean mechanization of a type theory with first-class universe levels, also based on @yiyunliu's [mltt-consistency](#) proof written in Rocq. This mechanization is closer in style than the one in Agda, since the logical relation takes advantage of Lean's impredicative Prop in place of induction–recursion. It has been checked with Lean 4.8.0-rc1 and requires Mathlib for some typeclasses. The development can be checked and built using `lake build`.

learning Lean

- Zulip (to ask questions): <https://leanprover.zulipchat.com/>
- Lean in your browser: <https://live.lean-lang.org/>
- reference manual: <https://lean-lang.org/lean4/doc/>
- library docs: https://leanprover-community.github.io/mathlib4_docs/
- learning resources: <https://leanprover-community.github.io/learn.html>

Books

- If you prefer reading a book (with exercises), the standard mathematics oriented reference is [Mathematics in Lean](#). You can also download it [as a pdf](#), but it is really meant to be used in VSCode, doing exercises on the fly (see the [instructions](#)).
- [The Mechanics of Proof](#) is also mathematics-oriented. It has a gentler pace than *Mathematics in Lean* and is aimed at readers with less mathematical experience.
- [Formalising Mathematics](#) is some course notes for mathematicians with useful tips and an overview of common tactics.
- If you prefer something more about the foundations of type theory, the standard reference is [Theorem Proving in Lean](#).
- A computer-science/programming oriented book is [The Hitchhiker's Guide to Logical Verification](#), which also has useful information about the type theory of Lean.