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:

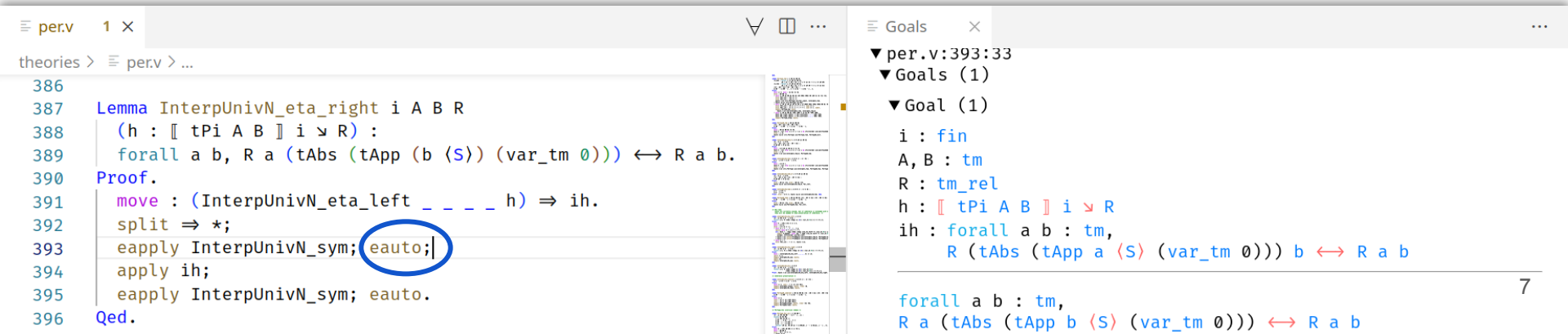
- Coq-LSP
- also applies to VSCoq 1/CoqIDE
- (I don't use emacs so idk Proof General)

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?

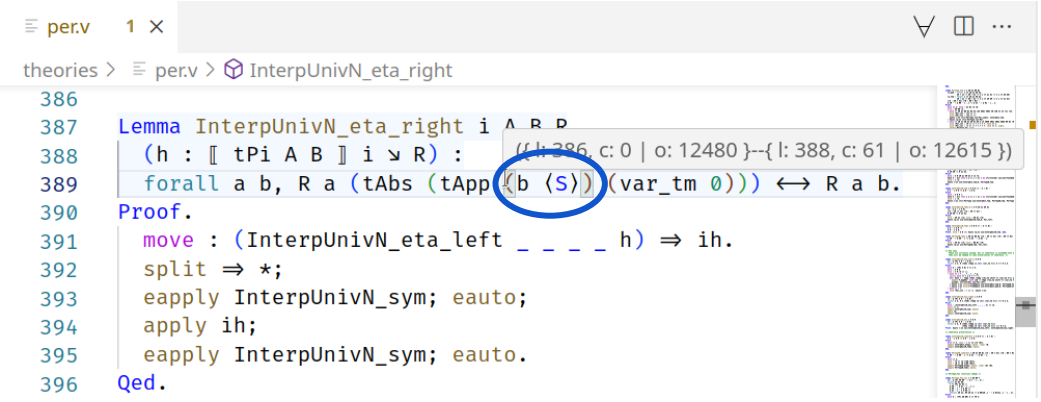
P1: proof reading is missing ephemeral information

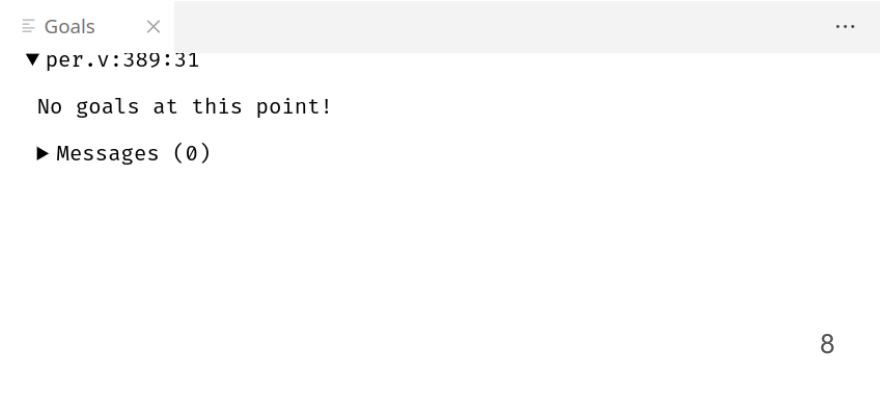
- Q1: what changes did this tactic make?
  - in `tac1; tac2`, what changes did tac1 make?
- Q2: what is the type of this subexpression?
  - what is the *context* of this subexpression?
- Q3: what case is this subproof for?



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?



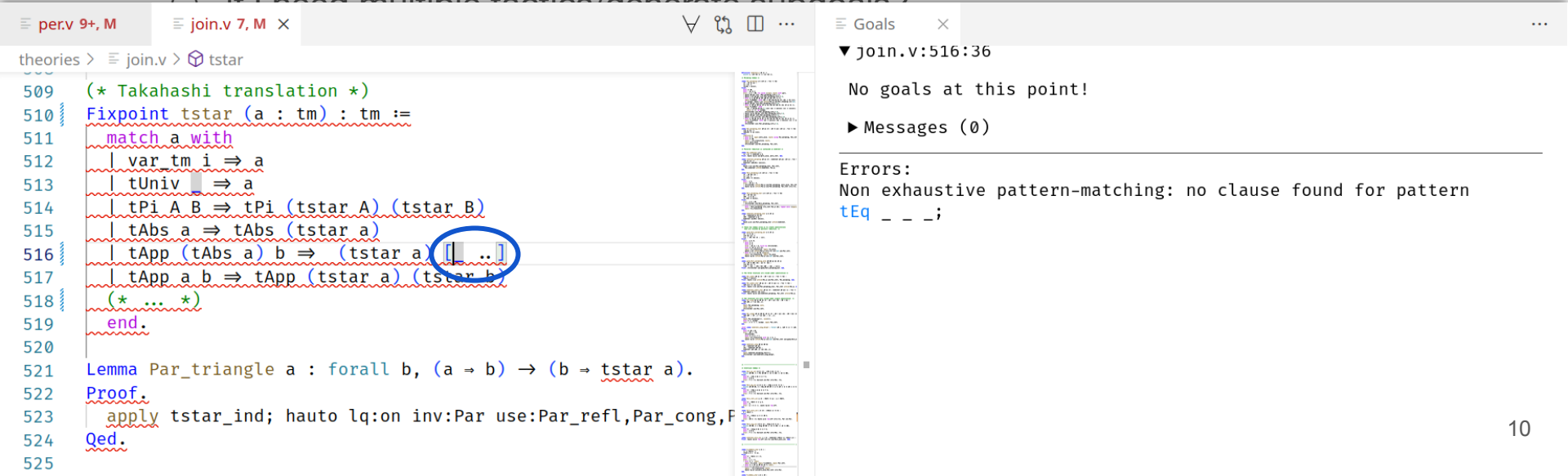


P2: term writing isn’t incremental

- 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?

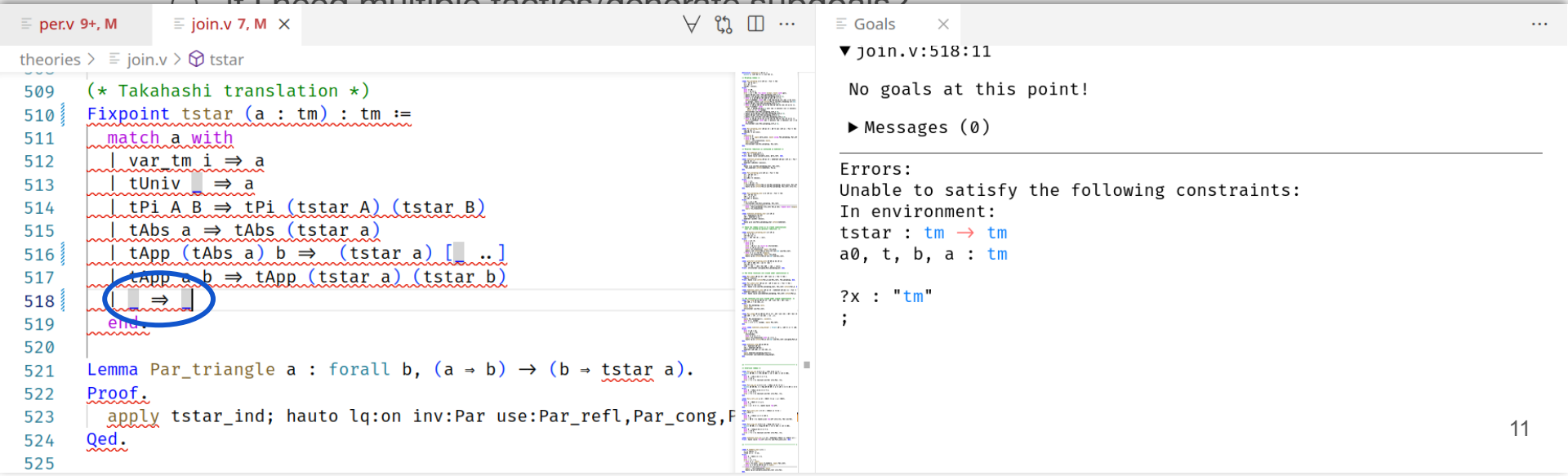
P2: term writing isn't incremental

- 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?



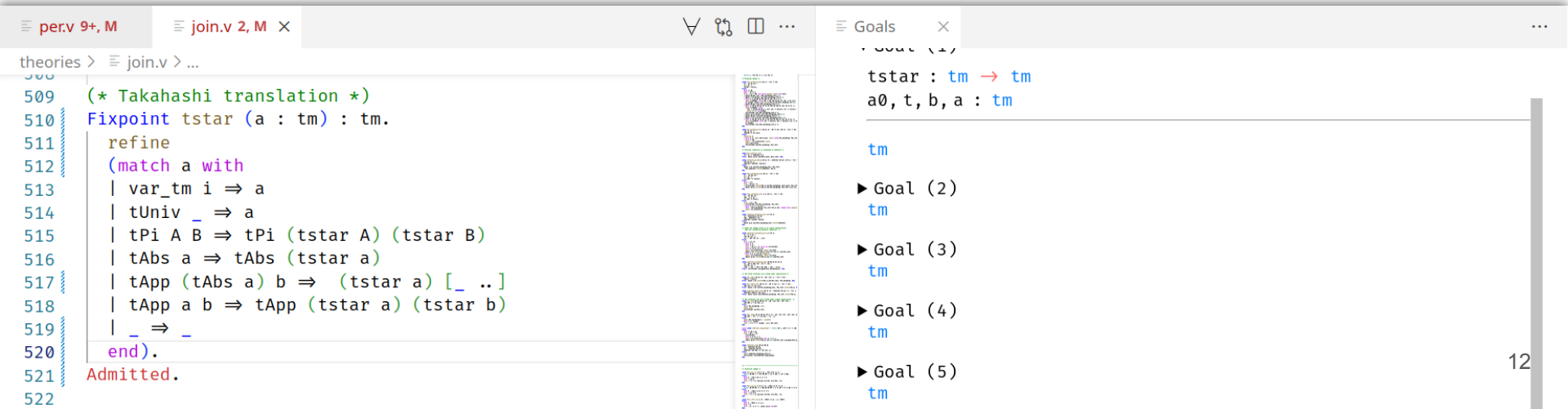
P2: term writing isn't incremental

- 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?

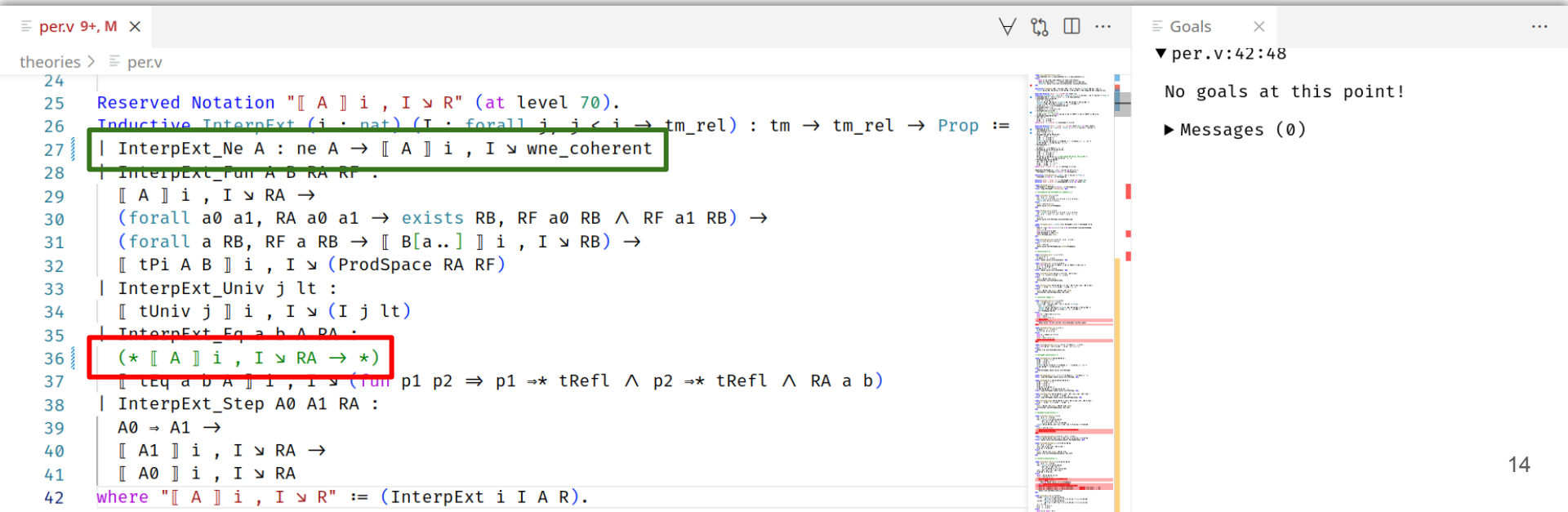


P3: proof editing is (more or less) sequential

- 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?

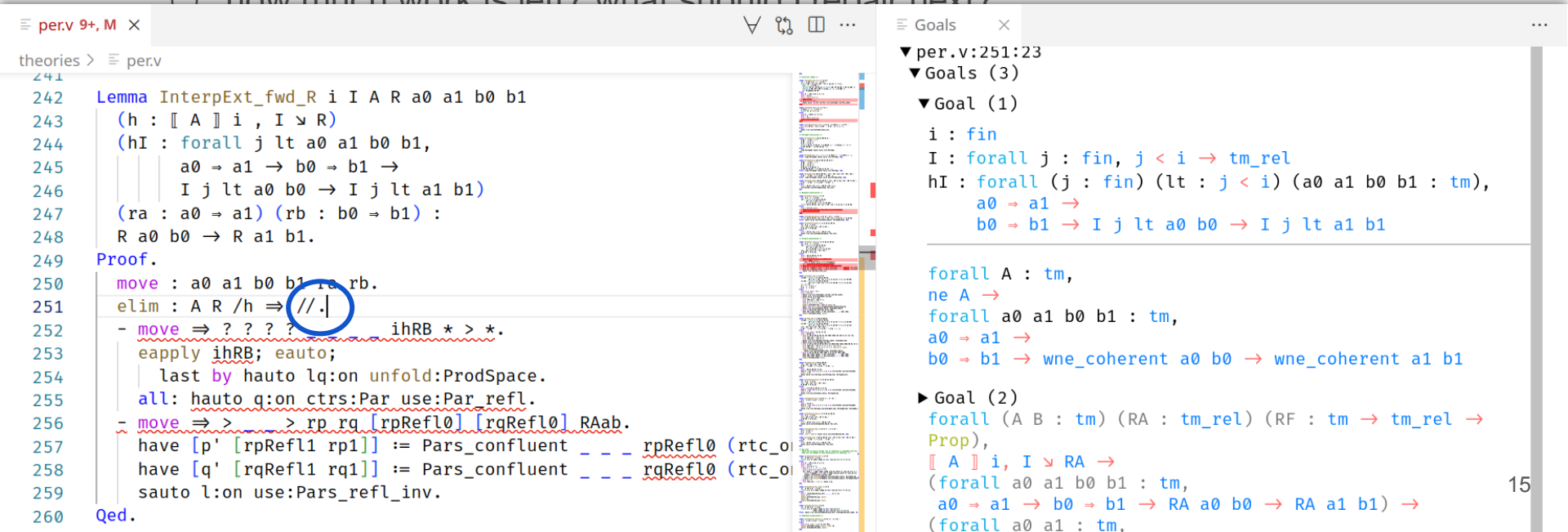
P3: proof editing is (more or less) sequential

- 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?



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?



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

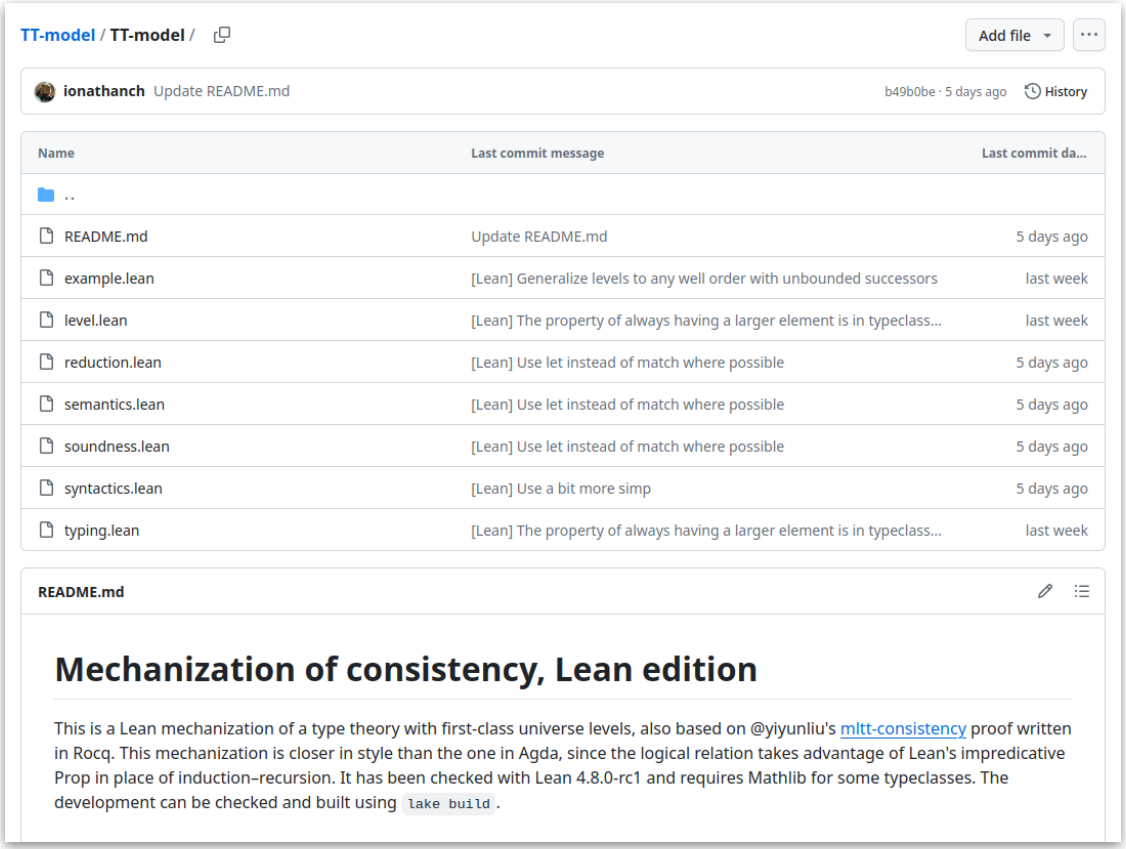
- LeanSSR
  https://github.com/verse-lab/lean-ssr
- Aesop
  https://github.com/leanprover-community/aesop
- Mathlib
  https://github.com/leanprover-community/mathlib4
- `(n)linarith` (part of Mathlib)

my issues with Lean

basically designed specifically targeting mathematics

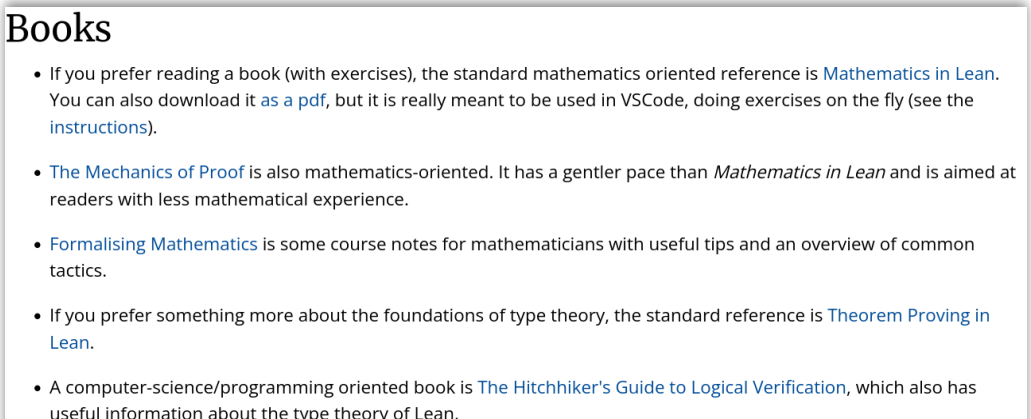
- no structural recursion on inductive predicates
  - inductive *propositions* are fine. smth abt decreasing measures
  - pretty sure this is a bug
- no induction tactic on mutual inductives
  - encode mutual inductives as single inductives
  - Coq doesn't have great mutual inductive support anyway
-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

ported existing development in ~1 week



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.