Elpi: rule-based meta-languge for Rocq

Enrico Tassi[1] - CoqPL 2025

1 Inria Centre at Université Côte d'Azur

This talk

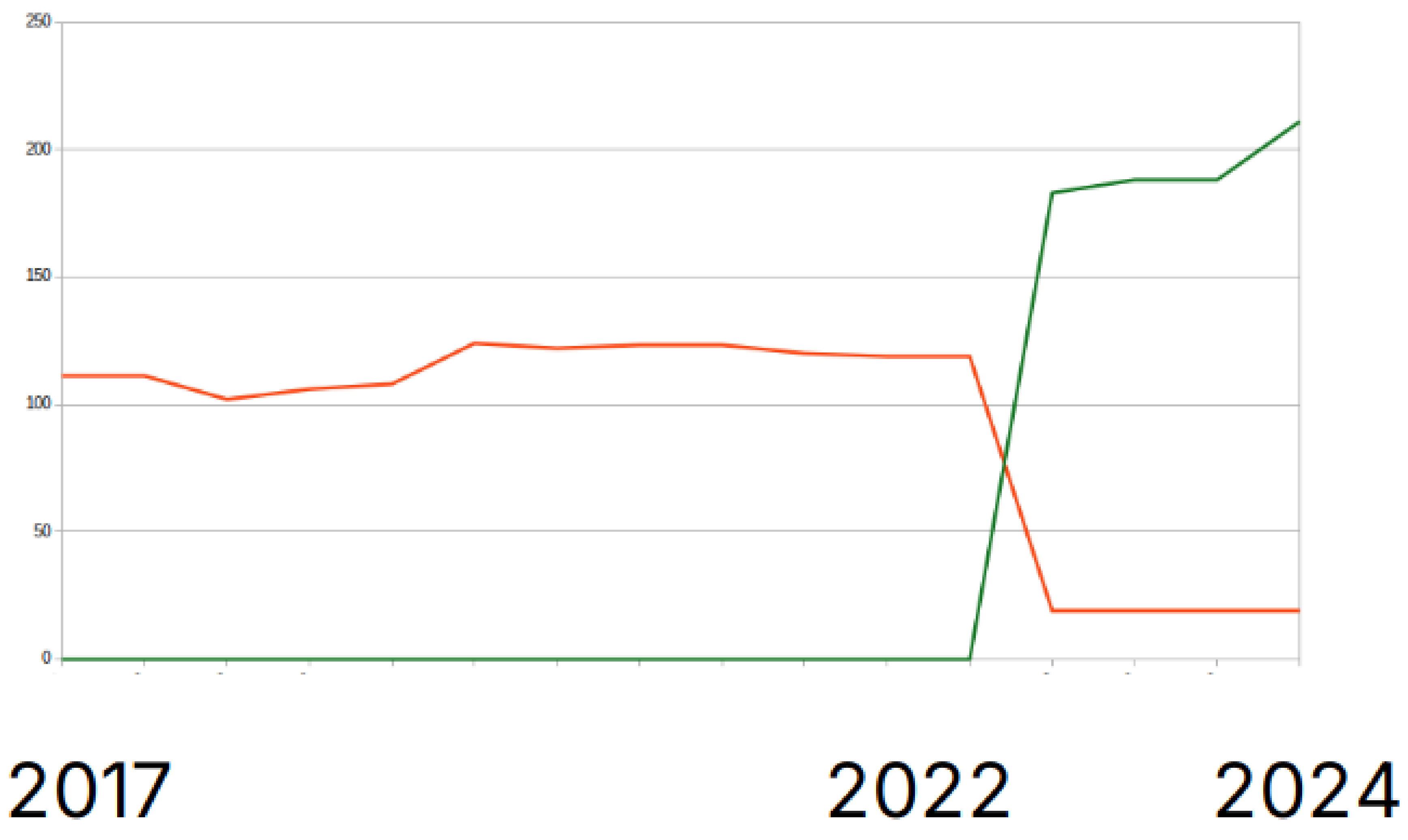
Users of Elpi

- https://github.com/math-comp/hierarchy-builder/
- https://github.com/coq-community/trocq
- https://github.com/LPCIC/coq-elpi/tree/master/apps/derive
- https://github.com/math-comp/algebra-tactics/

Hierarchy Builder

DSL to declare a hierarchy of interfaces

- generates boilerplate via Elpi's API: modules, implicit arguments, canonical structures, ...
- used by the Mathematical Components library and other ~20 libraries
- makes "packed classes" easy




```
From HB Require Import structures.

HB.mixin Record IsAddComoid A := {
  zero : A;
  add : A -> A -> A;
  addrA : forall x y z, add x (add y z) = add (add x y) z;
  addrC : forall x y, add x y = add y x;
  add0r : forall x, add zero x = x;
}.

HB.structure Definition AddComoid := { A of IsAddComoid A }.

Notation "0" := zero.
Infix "+" := add.

Check forall (M : AddComoid.type) (x : M), x + x = 0.
```

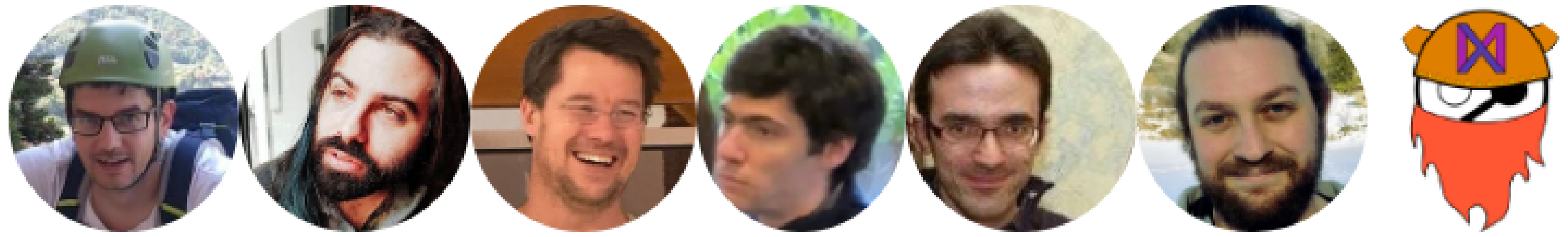
Trocq

Proof transfer via parametricity (with or without univalence).

- Registers in Elpi Databases translation rules
- Synthesizes transfer proofs minimizing the axioms required

```
From Trocq Require Import Trocq.

Definition RN : (N <=> nat)%P := ...
Trocq Use RN.

Lemma RN0 : RN 0%N 0%nat. ...
Lemma RNS m n : RN m n -> RN (N.succ m) (S n). ...
Trocq Use RN0 RNS.

Lemma N_Srec : ∀P : N -> Type, P 0%N ->
    (∀n, P n -> P (N.succ n)) -> ∀n, P n.
Proof.
trocq. (* replaces N by nat in the goal *)
exact nat_rect.
Qed.
```

Derive



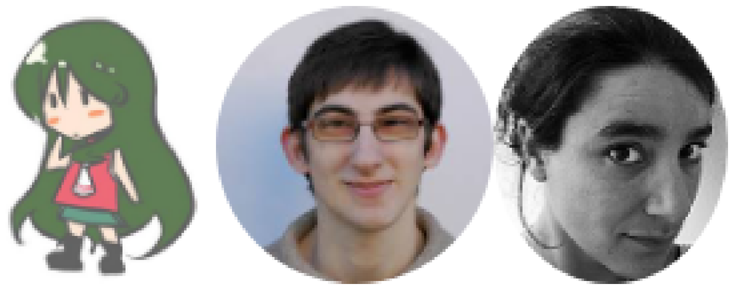
Framework for type driven code synthesis

Derivations:

- parametricity
- deep induction
- equality tests and proofs
- lenses (record update syntax)
- a few more...

```
From elpi.apps Require Import derive.std lens.

#[only(lens_laws, eqb), module] derive
Record Box A := { contents : A; tag : nat }.

About Box. (* Notation Box := Box.t *)

Check Box.eqb :
  ∀A, (A -> A -> bool) -> Box A -> Box A -> bool.

(* the Lens for the second field *)
Check @Box._tag : ∀A, Lens (Box A) (Box A) nat nat.

(* a Lens law *)
Check Box._tag_set_set : ∀A (r : Box A) y x,
  set Box._tag x (set Box._tag y r) = set Box._tag x r.
```

Algebra Tactics



`ring`, `field`, `lra`, `nra`, and `psatz` tactics for the Mathematical Components library.

- works with any instance of the structure: concrete, abstract and mixed like `int * R` where `R` is a variable
- automatically push down ring morphisms and additive functions to leaves of the expression
- reification up to instance unification in Elpi

```
From mathcomp Require Import all_ssreflect.
From mathcomp Require Import all_algebra.
From mathcomp Require Import ring lra.

Lemma test (F : realFieldType) (x y : F) :
  x + 2 * y <= 3 ->
  2 * x + y <= 3 ->
    x + y <= 2.
Proof. lra. Qed.

Variables (R : unitRingType) (x1 x2 x3 y1 y2 y3 : R).
Definition f1 : R := ...
Definition f2 : R := ...
Definition f3 : R := ...

(* 500 lines of polynomials later... *)

Lemma example_from_Sander : f1 * f2 = f3.
Proof. rewrite /f1 /f2 /f3. ring. Qed.
```

Elpi in a nutshell

https://github.com/LPCIC/elpi/

Rules, rules, rules!

Roots

- Elpi is a constraint logic programming language
- Elpi is a dialect of λProlog and CHR
- backtracking is not the point

⚠ Vintage syntax ahead

- variables are capitals `X`
- λx.t is written `x\ t`
- rules are written `goal :- subgoal1, subgoal2...`

What really matters

- Code is organized in rules
- Rule application is guided by a pattern
- Rules can be added statically and dynamically

Elpi: Hello World!

Simply typed λ-calculus in HOAS

```
kind tm type.
type app tm -> tm -> tm.
type lam (tm -> tm) -> tm.

kind ty type.
type arr ty -> ty -> ty.
```

Typing

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

```
goal> of (lam x\ lam y\ x) TyFst.
```

Elpi: Hello World!

Simply typed λ-calculus in HOAS

```
kind tm type.
type app tm -> tm -> tm.
type lam (tm -> tm) -> tm.

kind ty type.
type arr ty -> ty -> ty.
```

Typing

```
pred of i:tm, o:ty.
of c S0.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

```
goal> of (lam x\ lam y\ x) (arr S0 T0).
goal> of        (lam y\ c) T0.
```

Elpi: Hello World!

Simply typed λ-calculus in HOAS

```
kind tm type.
type app tm -> tm -> tm.
type lam (tm -> tm) -> tm.

kind ty type.
type arr ty -> ty -> ty.
```

Typing

```
pred of i:tm, o:ty.
of d S1.
of c S0.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

```
goal> of (lam x\ lam y\ x) (arr S0 (arr S1 T1)).
goal> of        (lam y\ c) (arr S1 T1).
goal> of                c  T1.
```

Elpi: Hello World!

Simply typed λ-calculus in HOAS

```
kind tm type.
type app tm -> tm -> tm.
type lam (tm -> tm) -> tm.

kind ty type.
type arr ty -> ty -> ty.
```

Typing

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

```
goal> of (lam x\ lam y\ x) TyFst.

Success:
  TyFst = arr S0 (arr S1 S0)
```

Elpi: Hello World!

Simply typed λ-calculus in HOAS

```
kind tm type.
type app tm -> tm -> tm.
type lam (tm -> tm) -> tm.

kind ty type.
type arr ty -> ty -> ty.
```

Typing

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

```
goal> of (app H A) T.

Failure.
```

Elpi = λProlog + CHR

Typing (as before)

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

Holes & constraints

```
of (uvar as E) T :-
  declare_constraint (of E T) [E].
```

```
goal> of (app H A) T.
```

Elpi = λProlog + CHR

Typing (as before)

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

Holes & constraints

```
of (uvar as E) T :-
  declare_constraint (of E T) [E].
```

```
goal> of (app H A) T.

Success:

Constraints:
  of A S  /* suspended on A */
  of H (arr S T)  /* suspended on H */
```

Elpi = λProlog + CHR

Typing (as before)

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

Holes & constraints

```
of (uvar as E) T :-
  declare_constraint (of E T) [E].
```

```
goal> of (app H A) T, H = (lam x\ x).

Success:

Constraints:
  of A T  /* suspended on A */
```

Elpi = λProlog + CHR

Typing (as before)

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

Holes & constraints

```
of (uvar as E) T :-
  declare_constraint (of E T) [E].
```

```
goal> of (app (lam x\ x) A) T.

Success:

Constraints:
  of A T  /* suspended on A */
```

Elpi = λProlog + CHR

Typing (as before)

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

Holes & constraints

```
of (uvar as E) T :-
  declare_constraint (of E T) [E].
```

```
goal> of (app D D) T.
```

Elpi = λProlog + CHR

Typing (as before)

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

Holes & constraints

```
of (uvar as E) T :-
  declare_constraint (of E T) [E].
```

```
goal> of (app D D) T.

Success:

Constraints:
  of D S  /* suspended on D */
  of D (arr S T)  /* suspended on D */
```

Elpi = λProlog + CHR

Typing (as before)

```
pred of i:tm, o:ty.
of (app H A) T :-
  of H (arr S T), of A S.
of (lam F) (arr S T) :-
  pi x\ of x S => of (F x) T.
```

Holes & constraints

```
of (uvar as E) T :-
  declare_constraint (of E T) [E].
```

Constraint Handling Rules

```
constraint of {
    rule (of X T1) \ (of X T2) <=> (T1 = T2).
}
```

```
goal> of (app D D) T.

Failure
```

Integration in Rocq

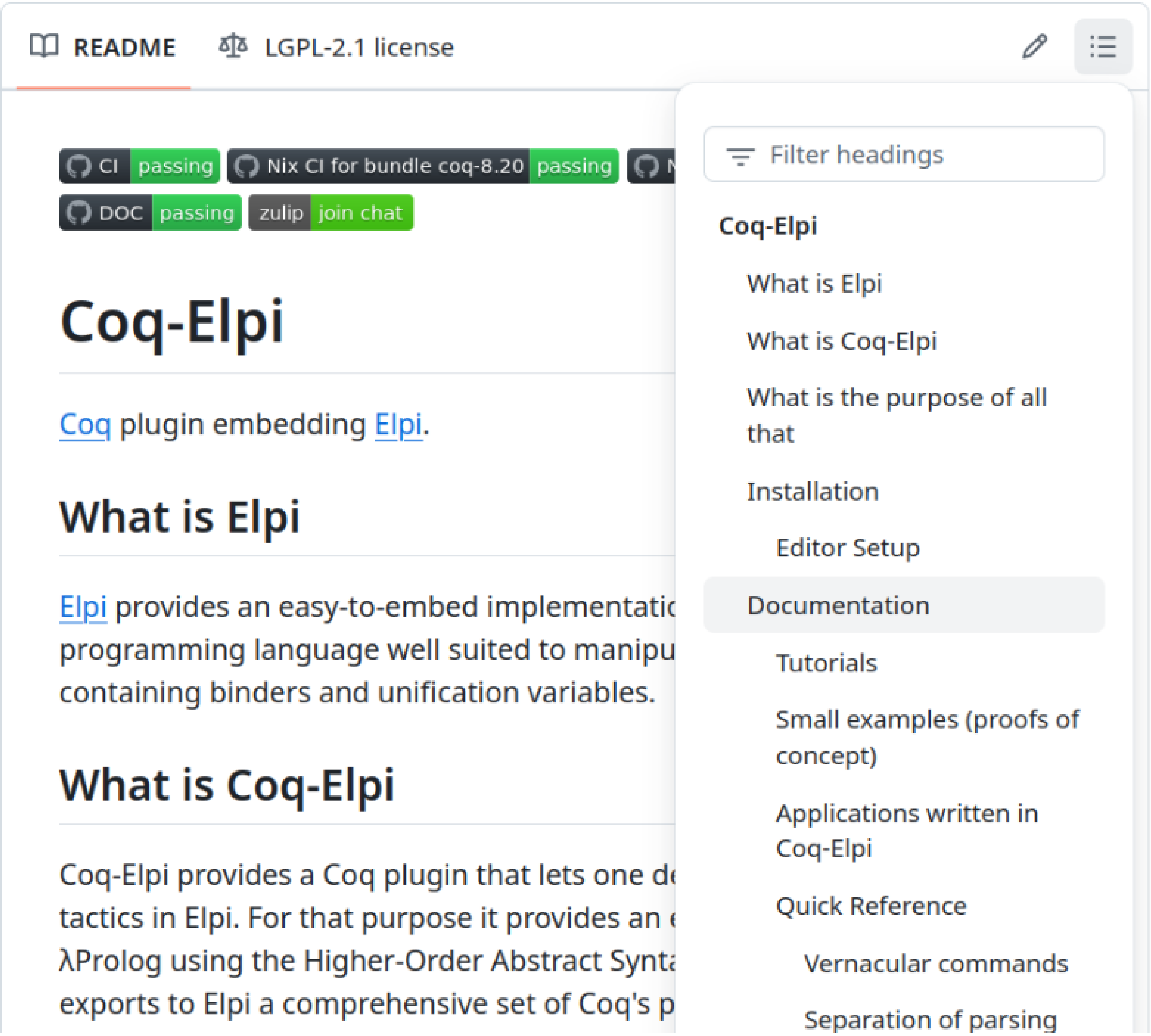
https://github.com/LPCIC/coq-elpi/

Notable features

- HOAS for Gallina
- quotations and anti-quotations

```
  coq.say {{ 1 + lp:{{ app[global S, {{ 0 }} ]  }}   }}
% elpi....  coq..     elpi..........  coq  elpi  coq
```

- Databases of rules
- Extensive API



Demo: from Prop to bool

```
Axiom is_even : nat -> Prop.

Fixpoint even n : bool := match n with
  | O => true
  | S (S n) => even n
  | _ => false
  end.

Lemma evenP n : reflect (is_even n) (even n).
(* Elpi add_tb evenP. *)

Lemma andP  {P Q : Prop} {p q : bool} :
  reflect P p -> reflect Q q ->
    reflect (P /\ Q) (p && q).
(* Elpi add_tb andP. *)

Lemma elimT {P b} : reflect P b -> b = true -> P.
```

```
Lemma test : is_even 6 /\ is_even 4.
Proof.
  refine (elimT (andP (evenP 6) (evenP 4)) _).
  (* elpi to_bool. *)
  simpl. trivial.
Qed.
```

```
(* [tb P R] finds R : reflect P _ *)
Elpi Tactic to_bool.
Elpi Accumulate lp:{{
  pred tb i:term, o:term.
  tb {{ is_even lp:N }} {{ evenP lp:N }}.
  tb {{ lp:P /\ lp:Q }} {{ andP lp:PP lp:QQ }} :- tb P PP, tb Q QQ.

  solve (goal _ _ Ty _ _ as G) GL :-
    tb Ty P, refine {{ elimT lp:P _ }} G GL.              }}.
```

Demo: from Prop to bool

```
Axiom is_even : nat -> Prop.

Fixpoint even n : bool := match n with
  | O => true
  | S (S n) => even n
  | _ => false
  end.

Lemma evenP n : reflect (is_even n) (even n).
(* Elpi add_tb evenP. *)

Lemma andP  {P Q : Prop} {p q : bool} :
  reflect P p -> reflect Q q ->
    reflect (P /\ Q) (p && q).
(* Elpi add_tb andP. *)

Lemma elimT {P b} : reflect P b -> b = true -> P.
```

```
Lemma test : is_even 6 /\ is_even 4.
Proof.
  refine (elimT (andP (evenP 6) (evenP 4)) _).
  (* elpi to_bool. *)
  simpl. trivial.
Qed.
```

```
(* [tb P R] finds R : reflect P _ *)
Elpi Db tb.db lp:{{ pred tb i:term, o:term. }}.

Elpi Tactic to_bool.
Elpi Accumulate Db tb.db.
Elpi Accumulate lp:{{
  solve (goal _ _ Ty _ _ as G) GL :-
    tb Ty P, refine {{ elimT lp:P _ }} G GL.           }}.

Elpi Command add_tb.
Elpi Accumulate Db tb.db.
Elpi Accumulate lp:{{
  pred compile i:term, i:term, i:list prop, o:prop.
  compile {{ reflect lp:P _ }} R Todo (tb P R :- Todo).
  compile {{ reflect lp:S _ -> lp:Ty }} R Todo (pi h\C h) :-
    pi h\ compile Ty {coq.mk-app R [h]} [tb S h|Todo] (C h).
  compile {{ forall x, lp:(Ty x) }} R Todo (pi x\ C x) :-
    pi x\ compile (Ty x) {coq.mk-app R [x]} Todo (C x).

  main [str S] :-
    coq.locate S GR,
    coq.env.typeof GR Ty,
    compile Ty (global GR) [] C,
    coq.elpi.accumulate _ "tb.db" (clause _ _ C).       }}.
```

The good company

https://github.com/coq-community/metaprogramming-rosetta-stone

Comparison

	Elpi	Ltac2	MetaCoq
Gallina	◕ no mutual fix/ind	●	●
Bound Variables	●	◔ quotations	◔ toplevel quotation
Holes	●	◕ tactic monad	◔ only AST
Proof API	◑ weak ltac1 bridge	● (sufficiently close)	◔ only TC search, obligations
Vernacular API	◕ no notations, obligations	○	◔ only env, obligations
Reasoning logic	◔ Abella	○	◕ no holes, unif

To the best of my knowledge, on 1/1/2025

Conclusion

Elpi for Rocq: take home

Extension language

- Use a language (ony) when it is a good fit
- Good FFI → many APIs!

Rule-based is a good fit for

- HOAS (binders and local context)
- prover logical environment (global context)
- (meta) meta programming (homoiconicity)

Ongoing and future work on Rocq-Elpi

- Type Class solver (D.Fissore PhD)
- Obligations (commands that start a proof)
- Mutual fixpoints and inductives (needed by 2 power users)

Ongoing and future work on Elpi

- Mode and determinacy analysis
- Memoization (tabling)

Thanks!

https://github.com/LPCIC/coq-elpi/

For having invited me, for listening, and for **contributing** code:

Pedro Abreu, Yves Bertot, Frederic Besson, Rob Blanco, Simon Boulier, Luc Chabassier, Cyril Cohen, Enzo Crance, Maxime Dénès, Jim Fehrle, Davide Fissore, Paolo G. Giarrusso, Gaëtan Gilbert, Benjamin Gregoire, Hugo Herbelin, Yoichi Hirai, Jasper Hugunin, Emilio Jesus Gallego Arias, Jan-Oliver Kaiser, Philip Kaludercic, Chantal Keller, Vincent Laporte, Jean-Christophe Léchenet, Rodolphe Lepigre, Karl Palmskog, Pierre-Marie Pédrot, Ramkumar Ramachandra, Pierre Roux, Pierre Roux, Claudio Sacerdoti Coen, Kazuhiko Sakaguchi, Matthieu Sozeau, Gordon Stewart, David Swasey, Alexey Trilis, Quentin Vermande, Théo Zimmermann, wdewe, whonore

Questions?