Towards typed-tactics in Coq: the what, the why, and the how

Beta Ziliani
CONICET / FAMAF - UNC

Joint work with **Jan-Oliver Kaiser, Yann Régis-Gianas, Robbert Krebbers**, with contribs from Béatrice Carré, Jacques-Pascal Deplaix, Thomas Refis.

WHAT is a tactic? #1: The Old Times

- A step in the elimination / introduction rules of the calculus:

intro x: $$\frac{x{:}P \vdash Q}{\vdash P \mathrel{\text{->}} Q}$$

- A *program* decomposing a goal into smaller *subgoals*:

apply lemma: $$\frac{\vdash P \qquad \vdash Q}{\vdash R}$$

lemma : P -> Q -> R

All written in OCaml

- A *program* to solve problems of a specific domain:

omega: $$\frac{}{x > 0 \mathrel{\text{->}} x + y > 0}$$

WHAT is a tactic?

#2 Ltac: A New View

- A *composition* of tactics (**ltac**):

```
eapply (tac_wp_pure _ _ _ _ (fill K e'));
 [ apply _                         (* PureExec *)
 | try fast_done                   (* The pure condition for PureExec *)
 | apply _                         (* IntoLaters *)
 | wp_expr_simpl_subst;            (* new goal *)
   try wp_value_head ]
```

(Snippet from the Iris project)

WHAT is a tactic? #2 Ltac (cont.)

- A (pretty weird) *functional program* manipulating *terms* and *goals* (**constr**):

```
Ltac of_expr e := lazymatch e with
  | heap_lang.Var ?x => constr:(Var x)
  | heap_lang.App ?e1 ?e2 =>
      let e1 := of_expr e1 in let e2 := of_expr e2 in constr:(App e1 e2)
  | _ => match goal with
          | H : Closed [] e |- _ => constr:(@ClosedExpr e H)
          end
  end.
```

(Snippet from the Iris project)

WHAT is a tactic?

#3 Ltac2: A Better Ltac

(Here Pim stands and sells Ltac2)

Problems with Ltac

- It's not a proper language:
  - It misses datatypes (e.g., no list for tactics),
  - Have no real typing (e.g., gets confused about **constr** and **ltac** in places it shouldn’t),
  - What is not provided can’t be coded (e.g., very limited support for goal reordering),
  - No proper error handling (e.g. just fail).
- Ltac2 improves the situation (!).

But there is one thing they still miss: **precise types in Gallina**!

WHY typed tactics? (ltac)

ARE THESE THE RIGHT NUMBER OF SUBGOALS?

```
eapply (tac_wp_pure _ _ _ _ _ (fill K e'));
  [ apply _                           (* PureExec *)
  | try fast_done                     (* The pure condition for PureExec *)
  | apply _                           (* IntoLaters *)
  | wp_expr_simpl_subst;              (* new goal *)
    try wp_value_head ]
```

ARE WE SHAPING THE NEW GOAL AS EXPECTED?

(Snippet from the Iris project)

WHY typed tactics? (constr)

ARE WE MISSING A CASE?

```
Ltac of_expr e := lazymatch e with
  | heap_lang.Var ?x => constr:(Var x)
  | heap_lang.App ?e1 ?e2 =>
      let e1 := of_expr e1 in let e2 := of_expr e2 in constr:(App e1 e2)
  | _ => match goal with
          | H : Closed [] e |- _ => constr:(@ClosedExpr e H)
          end
  end.
```

(Snippet from the Iris project)

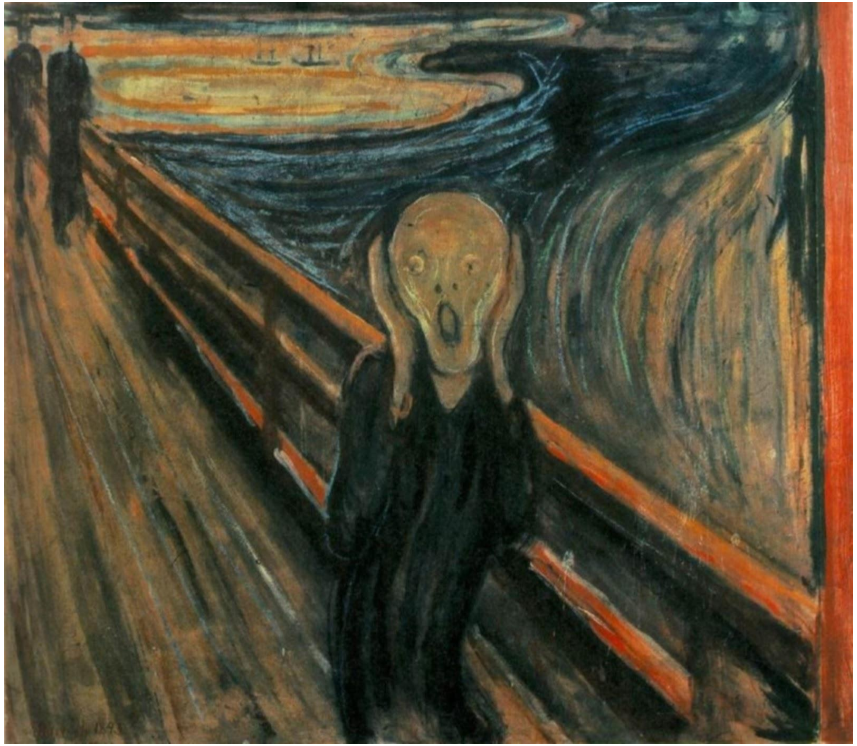
ARE THESE THE RIGHT ARGUMENTS?

ARE WE RETURNING THE RIGHT THING?

EVERY CASE CASES ON THE RIGHT TYPE?

A typo... a late-night change...

Set Ltac Debug.



Hypothesis

Types can help us obtain robust, maintainable tactics!

Typed tactics in Mtac2 (ltac)

THESE ARE THE RIGHT NUMBER OF SUBGOALS

```
`Δ' e2 φ <- M.evar _;
TT.apply (tac_wp_pure _ Δ' _ _ (fill K e') e2 φ _)
  <**> TT.by' T.apply_                  (* PureExec *)
  <**> TT.use (T.try fast_done)         (* The pure condition for PureExec *)
  <**> TT.by' T.apply_                  (* IntoLaters *)
  <**> (`e' <- M.evar _;                (* new goal *)
         wp_expr_simpl_subst e'
         <**> TT.try wp_value_head)
```

WE ARE SHAPING THE GOAL AS EXPECTED

Morally, <**> : (A -> B * goals) -> (A * goals) -> (B * goals)

Typed tactics in Mtac2 (constr)

WE ARE RETURNING THE RIGHT THING

```
Definition of_expr e  : heap_lang.expr → gtactic expr := mfix1 go e :=
  mtry
    match e with
    | heap_lang.Var x =>  T.ret (Var x)
    | heap_lang.App e1 e2 =>
         e1 <- go e1; e2 <- go e2; T.ret (App e1 e2)
    end
  with StuckTerm =>
     H <- T.select (Closed [] e); T.ret (@ClosedExpr e H)
  end.
```

THESE ARE THE RIGHT ARGUMENTS

WE CAN'T MISS A CASE

EVERY CASE CASES ON THE RIGHT TYPE

Mtac

A language for *typed* meta-programming (**constr**)

Typed meta-programs in Mtac (constr)

```
Definition of_expr e  : heap_lang.expr → M expr := mfix1 go e :=
  mtry
    match e with
    | heap_lang.Var x =>  ret (Var x)
    | heap_lang.App e1 e2 =>
        e1 <- go e1; e2 <- go e2; ret (App e1 e2)
    end
  with StuckTerm =>
     raise (WrongTerm e)
  end.
```

Meta-effects in the monad **M**

HOW we do meta-programming in Mtac

- Describe the "effects" in an inductive type **M**:

```
Inductive M : Type → Prop :=
| ret : A → M A
| bind : M A → (A → M B) → M B
| mtry : M A → (Exception → M A) → M A
| raise : Exception → M A
| mfix1 : ((∀x : A. M (B x)) → (∀x : A. M (B x)))→ ∀x : A. M (B x)
| ...
```

- Execute them in an interpreter.
    - It inherits β, δ, ι, ζ reductions from Coq.

The win of Mtac

- The typechecker catches errors at an early stage.
- A full-fledged functional language, with Coq's own stdlib, notation mechanism, etc.
- Undoubtedly better than Ltac's "constr:" [1].

[1] https://gmalecha.github.io/reflections/2016/04/18/experimenting-with-mtac/

Mtac2

Redesign of Mtac with support for tactic development (**ltac**)

Mtac2: Mtac + support for tactics (ltac and more)

Mtac +

1) A new proof environment MProof.
2) New language constructs: hypotheses, constrs, abs_let, ...
3) A *first-class* representation for goals within Coq.
4) (At the moment) two tactic types to describe two levels of correctness.
5) (Some) integration from-and-to Ltac.

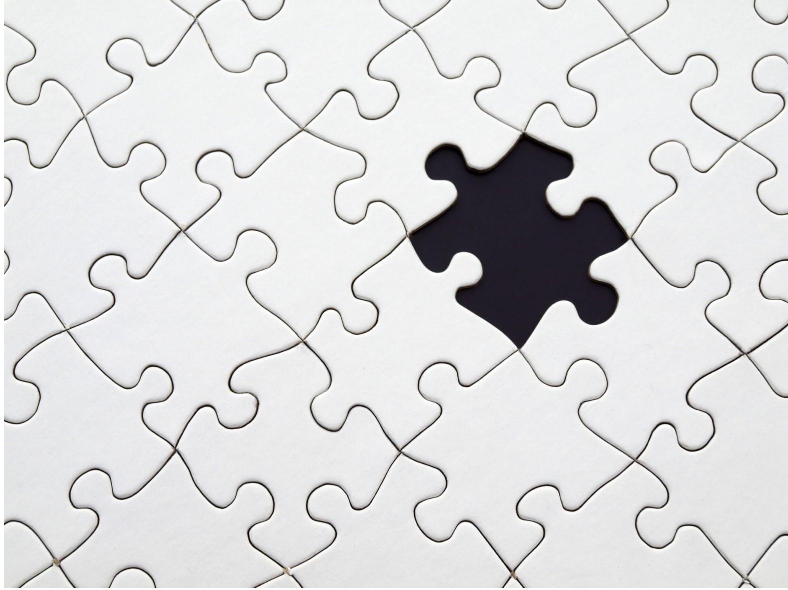
Use cases

1) First 6 files of Software Foundations
   a) To answer the question: do we have enough primitives to build tactics?
   b) Basic tactics: intros, destruct, intro patterns, apply, simpl, unfold, assert, generalize.
   c) Imported tactics from Ltac: inversion, induction, rewrite.

2) Several important tactics of Iris
   a) To answer the question: how can we juice out types for tactics?

Some challenges we faced

1) What is a good representation for **goals**?
2) What is a good representation for **tactics**?
3) How to avoid issues with **universes**?



What is a goal? (very partial answer)

- A goal is a *meta-variable*, but in Coq we just say is a term of some type:

```
Inductive goal :=
| Goal : forall {A}, A -> goal.
```

- However, different subgoals may have different contexts (demo).

Problem

How to compose tactics so that each work on the goal’s specific context

Solution: make goals carry their own context!

The Goal

A Hypothesis

What is a goal?   (partial yet sufficient answer)

```
Inductive goal :=
| Goal : forall {A}, A -> goal
| AHyp : forall {A}, (A -> goal) -> goal.

Theorem tl_length_pred : forall l: list nat,
  pred (length l) = length (tl l).
MProof.
  destructn 0 &> [m: idtac | intros n l ] &> reflexivity.
Qed.
```

```
[m: G ?x | AHyp (fun n=> AHyp (fun l => G ?y))]
```

What is a tactic? (untyped ltac fragment)

Considering a tactic as:

- A *program* decomposing a goal into smaller *subgoals* (apply).

Partial answer: a tactic takes a goal and returns a list of goals (in the **M** monad):

```
Definition tactic := goal -> M (list goal).
```

This is in essence the type of standard tactics (apply, intros, etc).

What is a tactic? (untyped ltac fragment)

- A *composition* of tactics (; operator in Ltac).

```
Class Seq (A : Type) :=
  &> : tactic -> A -> tactic.

Instance seq_one : Seq tactic := …

Instance seq_list : Seq (list tactic) := ...
```

What is a tactic? (constr fragment)

Now consider:

- A *functional program* manipulating *terms.*

A tactic takes a goal and returns a **value** and a list of goals (in the **M** monad):

```
Definition gtactic (A: Type) := goal -> M (A * list goal).
```

Unveiling the examples

```
Definition of_expr e  : heap_lang.expr → gtactic expr := mfix1 go e :=
 mtry
  match e with
  | heap_lang.Var x =>  T.ret (Var x)
  | heap_lang.App e1 e2 =>
     e1 <- go e1; e2 <- go e2; T.ret (App e1 e2)
  end
 with StuckTerm =>
   H <- T.select (Closed [] e); T.ret (@ClosedExpr e H)
 end.
```

T.ret a := ret (a, [])

SELECTS A HYPOTHESIS FROM THE GOAL

Unveiling the examples

```
`Δ’ e2 φ <- M.evar _;
TT.apply (tac_wp_pure _ Δ’ _ _ (fill K e’) e2 φ _)
  <**> TT.by’ T.apply_                        (* PureExec *)
  <**> TT.use (T.try fast_done)               (* The pure condition for PureExec *)
  <**> TT.by’ T.apply_                        (* IntoLaters *)
  <**> (`e’ <- M.evar _;                      (* new goal *)
          wp_expr_simpl_subst e’
          <**> TT.try wp_value_head)


Really,   <**> : M (A -> B * list goal) -> M (A * list goal) -> M (B * list goal)
```

Composition of tactics: combinatorial explosion!

```
intros &> T.select nat

apply x &> T.select nat

(a <**> b) &> [m: t1 | t2]
```

A universe of problems

Meta-programming for Coq in Coq

A universe of solutions

1) Universe polymorphism (UP).
2) Copy **list** and **prod** from std-lib.
   - Avoid interference of Mtac universes with user's.
   - Make them UP? Please?
3) **Avoid fixating universes at type M.**

Universes in Mtac

- The ***inductive type*** **M** with universe annotations:

```
Inductive M@{a b c d} : Type@{a} → Prop :=
  | ret : ∀ A : Type@{b}, A → M A
  | bind : ∀ (A : Type@{c}) (B : Type@{d}),
          M A → (A → M B) → M B
  | mtry : ∀ A : Type@{a},
          M A → (unit → M A) → M A
  | raise : ∀ A : Type@{a}, unit → M A
```

in which *b <= a , c <= a , d <= a*

SOME UNIVERSES ONLY APPEAR IN ONE CONSTRUCTOR, BUT THEY MUST ALL BE IN M

NECESSARY, YET RESTRICTIVE

Universes in Mtac2

- The ***inductive type*** **M** is just a type holder:

```
Inductive M@{a} : Type@{a} → Prop :=
  | mkM : ∀ A: Type@{a}, M A.
```

Opaque definition

```
Definition ret : ∀ A: Type@{c}, A →M A. … Qed.

Definition bind : ∀(A: Type@{d})(B: Type@{e}), M A → (A → M B) → M B. ...
Qed.
```

None of the universes are restricted!

What's missing in the picture? (Honest slide)

- Performance.
- Seriously, performance.
  - Getting much better playing with some cool ideas, but far from ideal.
  - Compilation?
- A serious study of universes (no idea how!).
- Reduce and GC universes: reflexivity has 520 universes!
  - Annotate universes is just too painful!
  - Not a real problem in the cases we studied, but it *feels* wrong.

Conclusions

- Types in tactics allow us to build maintainable tactics.
- Mtac2 provides a simple and integrated model for typed tactics.
- Tested in a real dev: Iris.
- Three challenges ahead: composition, performance and universes.
- Infinite possibilities for extensions.

M