Towards Ltac 2.0

P.-M. Pédrot

INRIA

DeepSpec Workshop

8th June 2016

How did we get there?

Tactics were introduced in Coq 4.10 (May 89).

```
Goal (a:A)(m:list) (Null (cons a m)).
Red.
Intros.
Do (resolve_unfolds) O_S.
Exact (length m).
Change <nat>(length nil)=(length (cons a m)).
Do (incomplet [3]) (f_equal length).
Assumption.
Save nil_cons.
```

Only primitive tactics at the time!

Ltac

Ltac was introduced in the 7.x branch by David Delahaye (around 2000).

Translated excerpt from the French `CHANGES` files:

> Ltac is a new layer of metalanguage to handle small automations.

Ltac

Ltac was introduced in the 7.x branch by David Delahaye (around 2000).

Translated excerpt from the French `CHANGES` files:

> Ltac is a new layer of metalanguage to handle small automations.

Ltac

Ltac was introduced in the 7.x branch by David Delahaye (around 2000).

Translated excerpt from the French `CHANGES` files:

> Ltac is a new layer of metalanguage to handle **small** automations.

Ltac

Ltac was introduced in the 7.x branch by David Delahaye (around 2000).

Translated excerpt from the French `CHANGES` files:

> Ltac is a new layer of metalanguage to handle **small** automations.

`<blink>SMALL</blink>`

That sentence never made it into the English documentation.

15 years later

2016: Probably more than 10^5 loc in Ltac (educated guess), which is:

- Not fitted for that scale
- Not specified
- Not specifiable
- Brittle
- Slow
- A kludgy pile of random stuff nobody really understands

« Ltac is the PHP of proof assistants »

A little issue

Facebook has billions of PHP loc, but if they were to start from scratch, they would probably use a decent language (Haskell, OCaml, ...).

We don't have that luck.

As of today, we don't know what a good tactic language is.

Experimental research: Mtac, Rtac, ssreflect...

Being Pragmatic

It looks like we can't really do much more about Ltac than in 2000.

Being Pragmatic

It looks like we can't really do much more about Ltac than in 2000.

WRONG!

Since Coq 8.5, we have a new tactic engine (A. Spiwack).

- Features backtrack and term refining
- Monadic ML API
- Ltac is built atop of it
- We can reason about the programs (hello, semantics!)

Being Pragmatic II

We can't make a perfect language, let us make a better Ltac.

Use the sane semantics of the tactic engine for Ltac 2.0

Ltac 2.0...

- will look like Ltac (syntax-wise)
- will kind of taste like Ltac (semantics-wise)
- but fortunately **won't be Ltac!** (implementation-wise)

The rough bluesheet

Following a very simple recipe:

1. Take the best minimalist language out there: ML
2. Interpret the ambient effects as the engine monad
3. Add some meta-programming facilities
4. Sprinkle notations here and there

... and *voilá*, you have Ltac 2.0.

Estimated implementation time for a prototype: 2 weeks.

What's for me in this?

In particular, Ltac 2.0 will be:

- typed
- supporting datatypes and programming features
- (partly) specified

Problems Ltac 2.0 won't solve:

- Unification
- Unspecified / unstable primitive tactics
- Efficiency (in a first time)
- Typing of metaprogramming

Acknowledging the existing

Thanks to notations, certain idioms should be mostly compatible, e.g.

```
intros [H|H]; destruct H as x; exact x.
```

Following guidelines, we can recover a certain amount of forward compatibility, e.g.

Don't write `let t := idtac; foo in … t …`
But write `let t _ := foo in … t () …`

TODO: write the guidelines!

(Un)luckily, crazy stuff is going to die in a horrible well-deserved suffering.

Transition path

Most probably, today's Ltac is going to survive through a plugin.

```
Require Import LegacyLtac.
```

This should content the crazy stuff people.

Other people would get a sane language when booting Coq.

Hopefully, crazy stuff is going to be rewritten at some point.

TODO

I am going to write the specs of Ltac 2.0 in a CEP.

Probably going to write a prototype as well at some point.

I had a dream

What if we fostered tactic DSL?