Notes on the Design of Standard Coq Tactics

Arthur Charguéraud

Coq meeting on tactics June 30th, 2009

Motivations

- Powerful for advanced users:
  - tactics that scale up well
  - tactics that are sufficiently robust
- Support different proof styles
  - everything can be named and traced
  - or names can be introduced automatically
- Easy to learn for beginners
  - a small and intuitive set of tactics
  - coherent behaviour and naming across tactics

Limit the number of tactics

– Use the most general tactic to eliminate redundancy:

  → apply behaves as eapply

  → clear behaves as dependent clear

– Use flags for fine-tuning of tactics

  → apply [simple] H

Can hope to drastically reduce the number of tactics (20?)

Different styles of "intros"

intros I1 .. IN

→ introduction of named arguments

introv I1 .. IN

→ name only the non-dependent hypotheses

intros.

→ introduction without naming any argument

The idea of "introv" can apply to destruct and induction.

```
Lemma demo_introv :
  forall a b, P1 a -> P2 b ->
  forall c d, P3 c -> P1 d -> c = b.
introv HA HB HC HD.
```

Naming of variables

Post-fix naming:

```
add E as I
destruct H as [H1 H2]
```

Pre-fix naming:

```
let I: E
let [H1 H2]: H
```

Both styles can be useful. How to duplicate the syntax?

N-ary operators

`split N, split N in H.`

→ split when it is a conjunction of N facts

`elim.`

→ if N-ary conjunctions is implemented with a type ProdN

Same applies for disjunction and existentials.

Integration of automation

Symbols `"~"` and `"*"` can follow any tactic.
→ they call tactic `auto_tilde` and `auto_star`, respectively
→ these tactics can be customized locally

For example:

```
Ltac auto_tilde := auto.
Ltac auto_star  := intuition eauto.
```

```
apply* (mylemma H).
assert~: (n > 0).
split~.
```

Advanced instantiation mode

The idea is to instantiate a lemma by giving only some of its arguments, and generating evar and subgoals for the one that are not provided. E1 is the lemma, and the other Ei describe its arguments (possibly underscores).

`let I: E1 .. EN.`
→ shorthand for **`let I: (instantiate E1 .. EN)`**

`apply E1 .. EN.`
→ shorthand for **`apply (instantiate E1 .. EN)`**

`specialize H E2 .. EN.`
→ intuitively, **`let H: (instantiate H E1 .. EN)`**

`forward I: E1 .. EN.`
→ intuitively, **`let I: E1 E2 EN _ _ _ _`**

Four instantiation modes

- **More "vars":** the user provides arguments for dependently-used variables, and subgoals are produced
- **Mode "hyps":** the user provides arguments for non-dependent hypotheses, and evars are produced
- **Mode "args":** the user provides all arguments
- **Mode "auto":** the user provides a mixture of variables and hypotheses, which are used on a first-match basis, meaning that underscores are inserted whenever needed.

Currently implemented with a hack in order to specify which instantiation mode should be used: if the first argument E1 provided is an instantiation mode, then this mode is used, otherwise the mode "auto" is used.