The `under` tactic (math-comp meeting)

Érik Martin-Dorel

Équipe ACADIE, Laboratoire IRIT
Université Toulouse III - Paul Sabatier

2019-02-28, Inria Sophia

Plan

Plan

Motivation

- Joint work with Sergei Soloviev
  - Formalizing Boolean games with random formulas as payoff functions (focus on random games, not on random "mixed" strategies)
  - `http://dx.doi.org/10.4230/LIPIcs.TYPES.2016.14`
  - Results include a formal proof of the probability that there is no winning strategy in whole classes of Boolean games
  - `https://github.com/erikmd/coq-bool-games`
- Proofs involving many manipulations of bigops
  - Development of a tactic in pure (Ltac1, `Tactic Notation`) to avoid (`evar`, `erewrite`)-bookkeeping when using `eq_bigr` (under) or `eq_bigl` (underp)
  - main file (`random_bool_games.v @ fad9bd6`) : 57 occurrences of `under`/`underp` for 600 LoC.Gallina + 1230 LoC.Proof

Generalization to single-condition eta lemmas

As suggested by Cyril :

- Generalize the tactic to be parameterized by the "eq_" lemma
- `https://github.com/erikmd/ssr-under-tac`

  ```
  (* Syntax, version 2 *)
  under [ssrpattern] eq_lemma [intropattern] tactic.
  under eq_lemma [intropattern] tactic.
  (* Exemples *)
  under [X in _ = X+_+_] eq_bigr [i Hi] rewrite GRing.mulrDl.
  under eq_bigr ? under eq_bigl ? rewrite setIT.
  ```

- Implementation still in pure Ltac1, with a couple of hacks.
- Limitation 1 : the `[ssrpattern]` cannot be `[in RHS]` because the term selection and the rewrite are uncoupled
- Limitation 2 : work only for lemmas with a particular structure (one single condition, a quantified equality, as last argument)

Reimplementation in OCaml

- Joint development with Enrico @ Coq Implementors Workshop 2018
- Tactics under and over.
- The previous 2 limitations are overcome.
- Applicable to any "Leibniz eta lemma" with 2 conditions (e.g., `eq_big`) or more.
- New syntax ; closer to math-comp style. . . (to be discussed)
- `https://github.com/coq/coq/pull/9651`

Plan

Syntax

- Interactive mode :

```
under vars: {occs}[patt]lemma.
- tac1 (* tweak the term under the binders *); over.
- tac2 (* tweak the term under the binders *); over.
...
```

- One-liner mode (currently implemented) :

```
under vars: {occs}[patt]lemma by tac1.
under vars: {occs}[patt]lemma by [tac1 | tac2].
```

- One-liner mode (latest proposal) :

```
under vars: {occs}[patt]lemma do tac1.
under vars: {occs}[patt]lemma do [tac1 | tac2].
```

Semantics

Tactic mostly useful for "Leibniz eta lemmas". Typical example :

```
Lemma example (P : nat -> bool) (F1 : nat -> nat) m :
  \sum_(0 <= i < m | P i) F1 i >= 0.
Proof. under i: eq_big do [tac1 | tac2].
```

1. Do `rewrite eq_big`, *without failing* but generating evars.
2. 3 subgoals are created (the side-conditions for the pred and the general term + the main subgoal)
3. For each subgoal created (except the main one), if its type is a product, it tries to introduce as many provided names as possible to the context (here, `move=> i`)
4. If the conclusion is a Leibniz eq. (e.g. `F1 i = ?Goal i`), it massages the goal to get the provably-equivalent goal (but *locked* w.r.t. done) `@Under _ (F1 i) (?Goal i)`, pretty-printed as `'Under[ F1 i ]`
5. Perform some dispatch applying `tac1; over`, etc. on the proper subgoals. (`over` : terminator instantiating the evar `?Goal`).
6. Do `simpl` on the only remaining main subgoal $\rightarrow$ no spurious β-redex.

More examples

- `[Demo]`

Discussion

- Coq PR : `https://github.com/coq/coq/pull/9651`
  - beyond replacing `by` with `do`, other things to do ?
- PR : `https://github.com/math-comp/math-comp/pull/292`
  - naming convention OK ?
    (`eq_mx`, `eq_poly`, `eq_ffun`, `eq_finset`, `eq_mktuple`)
  - could be shipped in 1.8.0 ?
- coq/coq#9651 is planned for Coq 8.10
  - "back-porting" (add support for other versions of Coq within math-comp) feasible ?