Unifying Unifiers

M. Sozeau
INRIA

Coq WG
October 4th 2017

Recap

- evars (ctx and type) and metas (no ctx)
- `unification.ml`, tactic unification (`apply`, `destruct`…):
  - `w_unify & abstract_list_all`
- `evarconv.ml`: refinement unification (`refine`, `Definition`…)
  - `evar_conv_x`

The plan & what happened

- Move everything to `evarconv.ml`
- `clenv` for evars already started by PMP, easy switch in general (`apply`, `auto`…), clean control on goals.
- Means moving higher-order abstraction to a new procedure: `second_order_matching`
- Along the way, fix bugs and fix specs.

Second-order matching

- Solve goals `?ev[σ] t1 .. tn ~= ty`
- In practice: everything (rewrite, destruct/elim/induction, set, apply…).
- clenv maintains the potentially HO clause (and the threading of sigmas is explicit, PMP would approve :).
- Rough idea: find the occurrences of t1 .. tn and σ in ty, abstract them by variables, instantiate ev by the predicate
- Subterm selection now always configurable: patterns, specific "equality" function, maximal well-typed generalization

The new evarconv

```
type unify_flags = {
  modulo_betaiota : bool;
  open_ts : Names.transparent_state;
  (* Reduction during unification *)
  closed_ts : Names.transparent_state;
  (* Transparency for closed terms conversion *)
  subterm_ts : Names.transparent_state;
  (* For subterm selection in HO unification *)
  frozen_evars : Evar.Set.t;
  allow_K_at_toplevel : bool
  (* Allow trivial HO unification solutions or not for explicit evar
arguments (in general, false) *) }
```

The new HO matching

This is *typed* generalisation of subterms, according to an occurrence selection parameter:

```
type occurrence_selection =
  | AtOccurrences of occurrences
  | Unspecified of prefer_abstraction
  (* choose abstraction over leaving the term if there is a choice *)

type occurrence_match_test =
  env -> evar_map -> int (* under binders *) ->    constr (* pattern *) -> constr (* subterm *) ->
  bool * evar_map

type occurrences_selection =
  occurrence_match_test * occurrence_selection list
  (** The occurrence selection list should be exactly of the length of the arguments of the existential
below *)

val second_order_matching : unify_flags -> env -> evar_map ->
  existential (* ev, σ, t1, .. tn *) -> occurrences_selection -> constr -> evar_map * bool
```

Status

- The standard library and test-suite compile
- For **apply**: **7** line change in the stdlib
    - Different instances chosen in non-linear unification, backward compat by specifying or making the script insensitive to that.
    - Bugfix apply which was shelving dependent subgoals, backward compatible with eapply.
    - Also, eapply was not shelving certain dependent subgoals, now fixed (incompatibilities in eauto).

Status: apply in test-suite

- Sometimes the user provides an explicit HO pattern, and uses a tactic such as elim which is supposed to do HO unification itself. We heuristically prefer FO unification in these cases. This should rather be an option/flag of apply/elim to prefer FO over HO.
- **with** bindings: unify their type with expected type before or after unifying with the conclusion?
  - Discrepancy between **apply** (before) and **exists** (after) here, solved by backtracking on failure to solve with bindings early (not ideal)

destruct/elim/inversion/…

- Not too hard to port, no incompatibilities spotted yet
- Allows occurrence selection (e.g. of indices), default is maximal abstraction (compatible).
- Time to fix a uniform order of side-conditions vs main subgoal (discrepancy between elim/destruct/induction)? Currently a parameter to clenv tactics.
- **Question:** Renaming trick done in `destruct`, do we really want this?

apply in auto/eauto/tc eauto

- Incompatibilities because apply works better (not failing to apply lemmas it should indeed be able to apply).
- Made treatment of transparent_state of databases consistent while I was at it, requiring a few directives.
- Need evaluation of impact here as well.

rewrite

- occurrence selection strategy: match subterms with same head as lhs and recursively matching applicative structure of explicit arguments (or explicit pattern, not necessary in stdlib, useful for switching to more or less conversion in specific cases).

- unification with delta following the applicative structure of the pattern (emulates a kind of FO approximation ensuring we respect what the user sees as “rigid” structure).

- Incompatibilities (~40 lines diff) old rewrite was arbitrarily reducing under constants in particular, or choosing the last occurrence instead of the first due to its FO heuristic, or choosing an arbitrary unifier.

rewrite: TODO

- bind occurrence selection strategy to ssrmatching strategy and see if there are differences.
- Evaluate incompatibilities on travis

Status

- Episode I - Infrastructure ready: https://github.com/coq/coq/pull/930 (evarconv and second_order_matching + reachable_from_evars test)
  - Small perf decrease in fiat-crypto certainly due to the later (need discussion with Hugo)
- Episode II - apply / eapply : https://github.com/coq/coq/pull/991
  - Need to adapt packages. Mainly apply’s which should rather be eapply’s AFAICS.

Status

- Episode III - the compatibility dilemma strikes back. Note: no nontrivial change in tactic API. I see two solutions for migrating:
  - Version switches: switch on global or local version in each modified tactic. Hard: where/how do we scope the versions?
  - Make the new tactics available as a "future" plugin in Coq, and so that Import Future binds to the new code (hopefully can be achieved, PMP?). That gives time for users to experiment and give feedback. If we're happy by 8.9 we switch the code. Might as well use this to cleanup other stuff.