Automatic proving in Coq

Yves Bertot

April 15, 2008

Combining tactics

- Then: ";" (apply the same tactic on all generated subgoals)
  - example: intros a b H; split; elim H; intros H H2; assumption
- Orelse: "| |"
  - example: intros a b H; elim H; intros H';
    (left || right); exact H'
  - span: ";[. . . |. . .]"
  - example: intros a b H; elim H;
    [intros H'; right; exact H' | intros H'; left; exact H']
- Neutral elements: idtac, fail.
- Repetition: repeat

Automatic proofs by simple repetition

- Remember: le_n : $\forall$n, n $\leq$ n
  le_S : $\forall$n, n $\leq$ S m
- Prove goals of the form S (...(S a)) = S (...(s a))
- repeat(apply le_n || apply le_S)

Automatic use of collections of theorems

- The tactic auto can be fine-tuned,
- This tactic repeatedly tries theorems taken from a database,
- Depth of repetition is limited drastically (default is 5 or 6),
- You add elements in databases by Hint Resolve *thm* : *db*.,
- You direct auto to use a given database *db* by typing auto with *db*,
- You change the depth by adding a number argument as in auto 20 with *db*,
- The tactic trivial is auto which refuses to use theorems with more than one premise.

Repeated use of rewriting theorems

- A tactic autorewrite repeats rewriting with collections of theorems,
- Example:

```
plus_assoc : forall n m p : nat, n + (m + p) = n + m + p
Hint Rewrite plus_assoc : assoc_db.
Lemma ex_autorw : ∀x y z t, x+((y+z)+t) = (x+y)+(z+t).
intros x y z t.
================
x + (y + z + t) = x + y + (z + t)
autorewrite with assoc_db.
================
x + y + z + t = x + y + z + t
trivial.
Qed.
```

Tactics for propositional logic

- The tactic intuition does more than auto, as it breaks the hypotheses down,
- The tactic intuition can be fine-tuned by writing an extra tactic expression behind,
- By default intuition calls auto with *
- Example:

```
Lemma ex_intuition :
∀A B x, 0 = S 0 \/ A /\ B → A /\ x ≤ x + x.
intuition (discriminate || auto with arith).
Qed.
```

Tactics for numbers

- ring will solve goals of the form $e_1 = e_2$,
  - if the two expressions are equal polynomials,
- ring_simplify will only simplify the equality,
- omega will solve goals containing comparisons between expressions,
  - if the members of all comparisons are linear formulas
- More powerful tactics are available with extra packages.
- ring and ring_simplify can be adapted to new ring structures.

Defining new tactics

- Ltac easy_compare := repeat(apply le_n || le_S).

Defining new tactics

- Ltac easy_compare := repeat(apply le_n || le_S).
- Defined tactics can take arguments,
- Ltac case_eq f :=
  generalize (refl_equal f); pattern f at -1; case f.

Adapting to the goal

- Pattern matching in tactics
  ```
  match goal with
    H : ?x = ?y |- ?x = ?z =>
      transitivity y;[exact H | idtac] end.
  ```
- You can also have pattern matching on expressions,
- Pattern-matching is not "functional programming" like,
  - Patterns don't need to be constructors,
  - Patterns don't need to be linear,
  - Pattern-matching backtracks upon failure.

Example of user-defined tactic

```
Ltac num_eq_list :=
 match goal with
  |- ?a::?b = ?c::?d =>
  let H := fresh in
  assert (H: a=c);[try ring; try omega |
   try rewrite H; clear H;
   assert (H : b = d);[apply refl_equal || eq_list |
                      try rewrite H; apply refl_equal]]
 end.
```

- This tactic will address goals of the form
  $a_1::a_2::a_3::\ldots = b_1::b_2::b_3::\ldots$
  and decompose it in goals
  $a_1{=}b_1 \qquad a_2{=}b_2 \qquad {}_3{=}b_3 \qquad \ldots$