An overview of the SSReflect extension to the Coq system

Anders Mörtberg

May 4, 2011

Background/History

- Extension to Coq
- George Gonthier: formalisation of the four color theorem
- Cayley-Hamilton theorem, decidability of ACF...
- Feit-Thompson theorem (part of the classification of finite simple groups)

ForMath project[1]

- ForMath - Formalisation of Mathematics:
  - Linear algebra
  - Algebraic topology and homological algebra
  - Real number computation and numerical analysis

[1] http://wiki.portal.chalmers.se/cse/pmwiki.php/ForMath/

Overview

- Library of mathematical theories
- “New” tactics and tacticals
- Small-scale reflection

Libraries

- $\sim$ 10000 proofs and 3000 definitions
- Boolean reflection, natural numbers (arithmetic, divisibility, gcd, prime decomposition), big operators, algebraic hierarchy, polynomials, linear algebra, group theory, etc...

“New” tactics/tacticals

- SSReflect scripts appear to divide evenly between:
  - Bookkeeping
  - Deduction
  - Rewriting
- The features are added not by adding new tactics but by extending the functionality of existing ones

Bookkeeping

- `=>` tactical moves "up"
- `:` tactical moves "down"
- Subsumes: `intros`, `generalize`, `rename`, `clear`, `pattern`

Deduction

- Bottom-up/Backward reasoning: `apply`
- Top-down/Forward reasoning: `have`, `suff`, `wlog`

Rewriting

- Extended `rewrite` tactic:
  - Rewrite both in any subset of the goal and context
  - Rewrites, simplifies, folding/unfolding, closing of subgoals
  - Chained rewriting
  - Pattern selection
- Subsumes: `rewrite`, `fold`, `unfold`

Example: rewrite

```
rewrite /my_def {2}[f _]/= my_eq //=.
```

- unfold my_def
- simplify second occurrence of pattern f _
- rewrite using my_eq
- simplify/close all generated subgoals

Small Scale Reflection

- Proof methodology
- Relate abstract representations to computable functions
- In ordinary reflection (e.g. the `ring` or `omega` tactics) the symbolic representation form is hidden

Example: $\leq$

```
Inductive leq n : nat -> Prop :=
  | leq_n : leq n n
  | leq_S : forall m, leq n m -> leq n (suc m).

Lemma leq_n_S :
  forall n m, leq n m -> leq (suc n) (suc m).
Proof.
intros n m n_leq_m.
elim n_leq_m.
  apply leq_n.
intros.
apply leq_S.
assumption.
Qed.
```

Example: $\leq$

```
Fixpoint sub n m := match n, m with
  | suc n', suc m' => sub n' m'
  | _, _ => n
  end.

Fixpoint eq n m := match n, m with
  | zero, zero => true
  | suc n', suc m' => eq n' m'
  | _, _ => false
  end.

Definition leq n m := eq (sub n m) zero.
```

Example: $\leq$

```
Lemma leq_n_S :
  forall n m, leq n m -> leq (suc n) (suc m).
Proof. by []. Qed.
```

```
Lemma leqn0 : forall n, (leq n zero) = (eq n zero).
Proof. by case. Qed.
```

Boolean reflection

"Prop and bool are truly complementary: the former supports robust natural deduction, the latter allows brute-force evaluation. SSReflect supplies a generic mechanism to have the best of the two worlds and move freely from a propositional version of a decidable predicate to its boolean version"

Example: Boolean reflection

```
Coercion is_true (b : bool) := b = true : Prop.

Inductive reflect (P : Prop) : bool -> Type :=
  | Reflect_true : P -> reflect P true
  | Reflect_false : ~ P -> reflect P false.

Lemma andP : reflect (b1 /\ b2) (b1 && b2).
Proof. by case b1; case b2; constructor=> //; case.
Qed.
```

Example: Boolean reflection

```
Variables T1 T2 : eqType.

Definition pair_eq (p q : T1 * T2) :=
  (p.1 == q.1) && (p.2 == q.2).

Lemma pair_eq1 :
  forall p q : T1 * T2, pair_eq p q -> p.1 == q.1.
Proof. by move=> [a b] [c d]; case/andP. Qed.
```

Questions?

This work has been partially funded by the FORMATH project, nr. 243847, of the FET program within the 7th Framework program of the European Commission