SSReflect 2

Jacques Garrigue, 2015 年 12 月 9 日

3　SSReflect のライブラリ

先週は ssreflect のコマンドを見たが、ssreflect の本当の強さはそのライブラリにある。その大きな特徴は書き換えを証明の基本手法とすること。

ライブラリは Ssreflect と MathComp という 2 つの部分からできている。前者は一般的なデータ構造で、後者は代数系の証明に使う。

Ssreflect

まず、Ssreflect を読み込む。それほど多くはない。

```
Require Import ssreflect ssrbool ssrnat ssrfun seq eqtype choice fintype.
```

ssrbool は論理式と述語の扱い。ssrnat は自然数。ssrfun は関数 (写像) の様々な性質。seq はリスト。eqtype，choice，fintype はそれぞれ等価性、選択、有限性が使えるデータ構造のための枠組みを提供している。例えば、自然数の等価性は判定できるので、排中律を仮定しなくても場合分けができる。

中身について、ファイルを参照するしかないが、まず ssrnat の例をみよう。(ちなみに、ソースファイルははは~/ssreflect の下にある。

```
Module Test_ssrnat.
Fixpoint sum n :=
  if n is m.+1 then n + sum m else 0.

Theorem sum_square n : 2 * sum n = n * n.+1.
Proof.
  elim: n => [|n IHn] /=.
    done.
  rewrite mulnDr.
  rewrite -[n.+2]addn2 mulnDr.
  rewrite [n.+1*n]mulnC -IHn.
  by rewrite addnC (mulnC _ 2).
Qed.
End Test_ssrnat.
```

自己反映

論理式も書き換えで処理したい。そのために、ssrbool では論理演算子を型 bool の上の演算子として定義している。例えば、/\ は&&, \/ は||になる。二つの定義の間に行き来するために、reflect という自己反映を表した宣言を使う。それが SSReflect の名前の由来である。

```
Print reflect.
Inductive reflect (P : Prop) : bool -> Set :=
    ReflectT : P -> reflect P true | ReflectF : ~ P -> reflect P false
Check orP.
orP : forall b1 b2 : bool, reflect (b1  b2) (b1 || b2)
```

表現の切り替えはビュー機構によって行われる。move, case, apply などの直後に/viewを付けると、対処が可能な方向に変換される。=>の右でも使える。なお、ビューとしては上の reflect 型だででなく、同値関係 (P <-> Q) や普通の定理 (P -> Q) も使える。

```
Module Test_ssrbool.
Variables a b c : bool.

Print andb.

Lemma andb_intro : a -> b -> a && b.
Proof.
  move=> a b.
  rewrite a.
  move=> /=.
  by rewrite b.
Qed.

Lemma andbC : a && b -> b && a.
Proof.
  case: a => /=.
    by rewrite andbT.
  done.
Restart.
  by case: a => //= ->.
Restart.
  by case: a; case: b.
Qed.

Lemma orbC : a || b -> b || a.
Proof.
  case: a => /=.
    by rewrite orbT.
  by rewrite orbF.
Restart.
  move/orP => H.
  apply/orP.
  move: H => [Ha|Hb].
    by right.
  by left.
Restart.
  by case: a; case: b.
Qed.
```

```
Lemma test_if x : if x == 3 then x*x == 9 else x !=3.
Proof.
  case Hx: (x == 3).
    by rewrite (eqP Hx).
  done.
Restart.
  case: ifP.
    by move/eqP ->.
  move/negbT. done.
Qed.

End Test_ssrbool.
```

MathComp

こちらはモジュールが多過ぎて、簡単に紹介できる。よく使うもおとして、finset(fintype に基いた有限集合、基本的な線形代数は matrix と perm、多項式は poly、素数は prime。

ここでは、初歩数論の div を応用した例をみる。大石君の提案で 2009 年度の京大の入試問題の一部である。

> a、b を互いに素な正の整数とし、a が奇数である。そのときに、$(a+b\sqrt{2})^n = a_n + b_n\sqrt{2}$ と定める。
>
> 1. a_2 は奇数であり、a_2 と b_2 は互いに素である。
> 2. 全ての n について、a_n は奇数であり、a_n と b_n は互いに素である。

とりあえず、(1) の一部を与える。

```
Require Import div.

Record ext2 : Set := Ext2 {pnat: nat; pirr: nat}.      (* 構造体の定義 *)

Definition prod2 (a b : ext2) :=
  Ext2 (pnat a*pnat b+2*pirr a*pirr b) (pnat a * pirr b + pirr a * pnat b).

Section Problem1.
Variables a b : nat.

Hypothesis odd_a : odd a.
Hypothesis Hab : coprime a b.

Definition ab := Ext2 a b.

Theorem odd_a2 : odd (pnat (prod2 ab ab)).
Proof.
  rewrite /prod2 /=.
  by rewrite odd_add odd_mul -mulnA mul2n odd_double addbC odd_a.
Qed.
```

```
Theorem mpr_a2 : coprime (pnat (prod2 ab ab)) (pirr (prod2 ab ab)).
Proof.
  rewrite /=.
  rewrite [b*a]mulnC addnn -muln2.
  rewrite !coprime_mulr /coprime.
  rewrite (gcdnC _ a) gcdnMDl.
  rewrite -/(coprime a (2 * b * b)) !coprime_mulr !Hab.
  rewrite coprimen2 odd_a /=.
Admitted.
```

練習問題 **3.1**　*1.* classic を使わずに以下を証明せよ。

```
Section Coq10.
Variables (A : Type) (P Q : A -> bool).

Lemma remove_c : forall (a : A),
  (forall x y, Q x = Q y) ->
  (forall c, ((exists x, P x) -> P c) -> Q c) -> Q a.
End Coq10.
```

2. 京大の問題を完成させよ。