Jacques Garrigue, 2017 年 11 月 29 日

MathComp/SSReflect

1 MathComp/SSReflect のインストール

講義のホームページから ssreflect-1.6-media.tgz をダウンロードし、それをホームディレクトリに展開すると使えるようになる。確認する方法は以下のとおり。

```
% cd ~
% tar zxvf ssreflect-1.6-media.tgz
...
% coqtop
Welcome to Coq 8.5

Coq < From mathcomp Require Import ssreflect.

Small Scale Reflection version 1.5 loaded.
Copyright 2005-2014 Microsoft Corporation and INRIA.
Distributed under the terms of the CeCILL-B license.
```

2 SSReflect のコマンド

SSReflect は 2 つの基本的な部分からできている。1 つは新しいコマンド。もう 1 つは新しい標準ライブラリ。

Coq に比べて、SSReflect の特徴は基本的に 7 つのコマンドのみで操作で証明が作れることだろう。

`move` 前提をゴールと仮定の間に移動させる。Coq の `intros`, `revert` や `generalize` に対応する。

`rewrite` ゴールや仮定の書き換えを行う。Coq の rewrite より強力になっていて、構文も少し違う。

`case` 場合分けを行う。Coq の `case` や `destruct` に対応する。

`have` 事実をおく。Coq の `assert` や `pose`, `set` に対応する。

`apply`, `elim` Coq の `apply` や `elim` と同じだが、SSReflect の修飾子が使える。

`by` ゴールの解決を強制する。

修飾子

これだけ少ないコマンドで済む理由は、各コマンドに修飾子を付けてその意味を強化できるからである。`move` がその代表的な例である。`move` は本来何もしないが、修飾子を自由に組み合わせることで様々な操作が可能である。まず、移動修飾子を見る。

`move => x y H.` ゴールの最初の 3 つの前提を仮定 `x y H` としておく。Coq の `intros x y H` と同じ。

`move: x H.` 仮定 `x` と `H` をゴールに戻す。`revert x H` と同じ。

move: (H). 仮定 H をゴールにコピーする (仮定として残す)。generalize H と同じ。

move: H => HP. 仮定 H 一端ゴールに戻して、HP という名前でもう一度おく。結果的に名前を H から HP に変えている。

上に書かれている「前提」という概念が SSReflect で重要な役割を果す。ゴールと区別され、名前のついている仮定に対して、前提はゴールに含まれ、名前があっても基本的には順番で参照される。

```
x : T
H1 : prop1
H2 : prop2
============================
forall y : T', prop3 -> prop4
```

ここでは二重線の上が仮定、下はゴールである。そして、T' が 1 つ目の前提、prop3 が 2 つ目の前提である。この前提は SSReflect にとってスタックの働きをしている。

分解パターンは destruct のような働きをする。書き換えパターンは前提でゴールを書き換える。評価修飾子は simpl や auto のような単純化や枝刈りを行う。どれも、=> の右側にしか書けない。

[x y] [Ha | Hb] これらは intros や destruct as で使えるものと同じ。最初の前提は対として分解し、中身を x と y にする。次の前提について場合分けを行う。

-> <- 最初の前提で残りのゴールを書き換える。前提が消える。intros H; rewrite H か<- H と同じ。

/= ゴールを評価する。simpl と同じ。

// 自明な場合を省く。後で見る try done と同じ。

//= 両方を同時に行う。

case, apply, elim で上記の全ての修飾子が使える。rewrite や Coq 本来のコマンドでは、=> 以降のものしか使えない。例えば n : nat で、Hn が n に依存したとき、

```
elim: n Hn => /= [|n IHn] // ->.
```

これはまず n と Hn をゴールに戻し、最初の前提となる n に対して帰納法を使う。得られたゴールを/=で単純化し、n が 0 でないとき、新しい前提ができるので、それらを n と IHn として仮定に加える。次に自明な場合を省き、最後に Hn で残りを書き換える。

rewrite

rewrite は独自の構文を使う。基本的には、定理の名前や適用を並べる。

```
rewrite lem1 lem2 (lem3 n 1)
```

各定理に対して、繰り返しや適用箇所の指定もできる。

!lem 定理 lem による書き換えを可能な限り繰り返す。

2!lem 定理 lem による書き換えを 2 回繰り返す。

?lem 定理 lem を 0 または一回使う。

-lem 定理 lem を左向きに使う。

{2}lem 2 番目の出現を書き換える

[_ + n]lem パターン $(\_ + n)$ にマッチする最初の出現を書き換える。

/def 定義 def を展開する。unfold と同じ。

-/def 定義 def を畳み込む。fold と同じ。

上記の書き換え修飾子を組合せることができるが、順番に気を付けなければならない。

```
rewrite -{2}[_ + n]lem
```

また、定理や定義の間に評価修飾子を (/=, //, //=) を挿入してもいい。

have, case, by, done

have には 2 つの形がある。

have H : prop. 命題 prop の証明を始める。証明が終ったらら仮定 H として加える。assert と同じ

have H := lem arg1 arg2. 定理 lem を適用したものを仮定 H として加える。pose と同じ。

どちらの場合でも、H が省略されると、命題が前提として置かれる。また、H の代わりに分解パターンや書き換えパターンを使ってもいい。

```
have [x Hx]: exists x, x * x = 9.
```

case にも特殊な構文がある。

case H: x. 場合分けに関する等式を仮定 H としておく。case_eq に似ている。

最後に by と done を見る。

by cmds. コマンド列 cmds の後に auto の弱い形でゴールを解決する。解決できない場合、エラーになる。

done. by [] と同じ。この時点で解決できなければエラーになる。

例

```
From mathcomp Require Import ssreflect.

Section Koushin.
Variables P Q : Prop.

Theorem modus_ponens : P -> (P -> Q) -> Q.
Proof.
  by move=> p; apply.
Qed.

Theorem DeMorgan : ~ (P \/ Q) -> ~ P /\ ~ Q.
Proof.
  move=> npq.
  by split=> [p|q]; apply npq; [left | right].
Qed.
```

```
Hypothesis classic : forall P, ~~P -> P.

Theorem DeMorgan' : ~ (P /\ Q) -> ~ P \/ ~ Q.
Proof.
  move=> npq.
  apply classic => nnpq.
  apply: npq.  (* npq を move してから apply *)
  split; apply classic => n.
    by apply nnpq; left.
  by apply nnpq; right.
Qed.

Theorem and_comm : P /\ Q -> Q /\ P.
Proof.
  by move=> [p q]; split.
Qed.
End Koushin.

Module Plus.
Lemma plus_assoc m n p : m + (n + p) = (m + n) + p.
Proof.
  elim: m => [|m IHm] //=.
  by rewrite IHm.
Qed.

Require Import Arith.

Theorem avg_prod m n p : m+n = p+p -> (p - n) * (p - m) = 0.
Proof.
  move=> Hmn.
  have Hp0: forall q, p <= q -> p-q = 0.
    move=> q Hq.
    have Hpq := minus_le_compat_r _ _ q Hq.
    rewrite minus_diag in Hpq.
    by rewrite -(le_n_0_eq _ Hpq).
  have [Hpm|Hpn]: (p <= m) \/ (p <= n).
    case: (le_lt_dec p m) => Hpm.
      by left.
    case: (le_lt_dec p n) => Hpn.
      by right.
    have Hbad: m + n < p + p.
      by apply plus_lt_compat.
    rewrite Hmn in Hbad.
    by elim: (lt_irrefl _ Hbad).
   by rewrite (Hp0 _ Hpm) mult_0_r.
  by rewrite (Hp0 _ Hpn) mult_0_l.
Qed.
End Plus.
```

練習問題 2.1 今までの課題を *SSReflect* の構文を使って書き換えてみる。