Jacques Garrigue, 2021 年 11 月 10 日

Mathcomp, 自己反映と数論の証明

1 MathComp のライブラリ

先週は ssreflect のコマンドを見たが，MathComp 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`, `fingroup`, `algebra` 等，いくつかの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．

Search

Ssreflect の Search コマンドが強力で，ライブラリを探すのに便利.

```
Search "add".                                  (* 名前に add を含む定理を検索する *)
Search (_ + S _).                          (* 結論がパターンを含む定理を検索する *)
Search _ (_ + S _).                (* 前提または結論がパターンを含む定理を検索する *)
Search (_ + _) (_ * _) "mul".                  (* 左を全てみたすものを検索する *)
```

基本データ

まず，`ssreflect` を読み込む.[1]

```
From mathcomp Require Import all_ssreflect.
```

いくつかのモジュールが読み込まれます．`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.[2]

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

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

[1] もしも mathcomp がまだインストールされていなければ，講義のホームページからダウンロードして展開する．

[2] MathComp/ssreflect のソースファイルはメディアラボの場合は `~/.local/share/coq/mathcomp/ssreflect` と `/opt/local/lib/coq/theories/ssr` の下にあり，Coq Platform の場合，そのリソースの `lib/coq/theories/ssr` と `lib/coq/user-contrib/mathcomp/ssreflect`. Coq Platform のリソース自体は MacOS の場合は `/Applications/Coq_Platform_2021.02.1.app/Contents/Resources` などにある．

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．前に見た適用パターンを使う．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=> /=.
  done.
Restart.
  by move ->.
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.
```

自己反映があると自然数の証明もスムーズになる.

```
Theorem avg_prod2 m n p : m+n = p+p -> (p - n) * (p - m) = 0.
Proof.
  move=> Hmn.
  have Hp0 q: p <= q -> p-q = 0.
    rewrite -subn_eq0. by move/eqP.
  suff /orP[Hpm|Hpn]: (p <= m) || (p <= n).
    - by rewrite (Hp0 m) // muln0.
    - by rewrite (Hp0 n).
  case: (leqP p m) => Hpm //=.
  case: (leqP p n) => Hpn //=.
  suff: m + n < p + p.
    by rewrite Hmn ltnn.
  by rewrite -addnS leq_add // ltnW.
Qed.
```

練習問題 1.1 以下の等式を証明しなさい．タクティクは rewrite のみでできる．
ssrnat_doc.v の補題でほぼ足りるが, leq_mul も便利.

```
Module Equalities.
  Theorem square_sum a b : (a + b)^2 = a^2 + 2 * a * b + b^2. Abort.
  Theorem diff_square m n : m >= n -> m^2 - n^2 = (m+n) * (m-n). Abort.
  Theorem square_diff m n : m >= n -> (m-n)^2 = m^2 + n^2 - 2 * m * n. Abort.
End Equalities.
```

2 最大公約数の計算

ユークリッドが発明した互除法による最大公約数の計算は多分世界最古のアルゴリズムの一つである．その正しさを証明する.

```
let rec gcd m n =
  if m = 0 then n else gcd (n mod m) m
```

最大公約数の厳密な定義は

$$q \text{ は } m \text{ と } n \text{ の最大公約数である} \Leftrightarrow \begin{cases} q \mid m \ \wedge \ q \mid n \\ \forall q', (q' \mid m \ \wedge \ q' \mid n) \Rightarrow q' \mid q \end{cases}$$

二つ目に関して，本来は $q' \leq q$ のはずだが，$q' \mid q$ の方が証明しやすい．証明は m に関する簡単な帰納法である．

上の定義を Coq に与えると問題が生じる．

```
Fail Fixpoint gcd (m n : nat) {struct m} : nat :=
  if m is 0 then n else gcd (n %% m) m.
Error:
Recursive definition of gcd is ill-formed.
Recursive call to gcd has principal argument equal to
"n %% m" instead of "n0".
```

どうも，Coq が n %% m が m より小さいことを理解していないようだ．解決法は 2 つある．

ダミーの引数　常に m より大きいダミーの引数を追加して，その引数に対する帰納法を使う．

```
Fixpoint gcd (h m n : nat) {struct h} : nat :=
  if h is h.+1 then
    if m is 0 then n else gcd h (n %% m) m
  else 0.
```

h に関する場合分けが常に成功する（h が 0 になることはない）ことを証明しなければならないが，難しくはない．しかし，このやり方を使うと，Extraction の後でも h がコードの中に残り，本来のアルゴリズムと少し違ってしまう．

整礎帰納法　整礎な順序とは，無限な減少列を持たない順序のことを言う．自然数の上では $<$ は整礎である．特定の引数が全ての再帰呼び出しで整礎な順序において減少しているならば，関数の計算が無限に続くことはないので，Coq が定義を認める．(実際には減少の証明の構造に関する構造的帰納法が使われている)

Fixpoint の代わりに Function を使い，struct（構造）を wf（整礎）に変える．この方法では，定義と同時に引数が小さくなることを証明しなければならない．

```
Require Import Wf_nat Recdef.
Check lt_wf.
     : well_founded lt
Check lt_wf_ind.
     : ∀n (P : nat → Prop), (∀n', (∀m, m < n' → P m) → P n') → P n

Function gcd (m n : nat) {wf lt m} : nat :=
  if m is 0 then n else gcd (modn n m) m.
Proof.
- move=> m n m0 _. apply/ltP.
  by rewrite ltn_mod.
- exact: lt_wf.
Qed.
gcd_ind is defined
...
gcd is defined
gcd_equation is defined
Check gcd_equation.
Check gcd_ind.
Print gcd_terminate.

Require Import Extraction.
Extraction gcd.                                    (* wf が消える *)
```

```
let rec gcd m n =
  match m with
  | O -> n
  | S n0 -> gcd (modn n (S n0)) (S n0)
```

では，これから正しさを証明する．

```
Search (_ %| _) "dvdn".                    (* 割り切ることに関する補題を表示 *)
Check divn_eq.
     : ∀ m d : nat, m = m %/ d * d + m %% d

Theorem gcd_divides m n : (gcd m n %| m) && (gcd m n %| n).
Proof.
  functional induction (gcd m n).
    by rewrite dvdn0 dvdnn.
Admitted.

Check addKn.
     : ∀ x y : nat, x + y - x = y

Theorem gcd_max g m n : g %| m -> g %| n -> g %| gcd m n.
Admitted.
```

練習問題 2.1 `Admitted` を `Qed` に変え，証明を完成せよ．

3 $\sqrt{2}$ が無理数

まずは自然数で以下の定理を証明する．

定理 1 任意の自然数 n と p について，

$$n \cdot n = 2(p \cdot p) \text{ ならば } p = 0$$

証明は n の関する整礎帰納法を使う．

- $n = 0$ のとき，$p = 0$
- $n \neq 0$ のとき，
  - n と p が偶数でなければならないので，$n = 2n', p = 2p'$ とおける
  - 再び，$n' \cdot n' = 2(p' \cdot p')$ が得られ，$n' < n$
  - 帰納法の仮定より $p' = 0$
  - すなわち，$p = 0$

その定理を使って，$\sqrt{2}$ が無理数であることを証明する．もしも $\sqrt{2}$ が有理数なら，ある n と p が存在し，$\sqrt{2} = n/p$，すなわち $n^2 = 2p^2$．しかし上の定理から $p = 0$ となるので矛盾．

実際に証明する．

```
odd_mul    : ∀ m n : nat, odd (m * n) = odd m && odd n
odd_double : ∀ n : nat, odd n.*2 = false
odd_double_half : ∀ n : nat, odd n + (n./2).*2 = n
andbb      : ∀ x : bool, x && x = x
negbTE     : ∀ b : bool, ~~ b -> b = false
double_inj : ∀ x x2 : nat, x.*2 = x2.*2 -> x = x2
```

```
divn2      : ∀ m : nat, m %/ 2 = m./2
ltn_Pdiv   : ∀ m d : nat, 1 < d -> 0 < m -> m %/ d < m
muln2      : ∀ m : nat, m * 2 = m.*2
esym       : ∀ (A : Type) (x y : A), x = y -> y = x

Lemma odd_square n : odd n = odd (n*n). Admitted.
Lemma even_double_half n : ~~odd n -> n./2.*2 = n. Admitted.


(* 本定理 *)
Theorem main_thm (n p : nat) : n * n = (p * p).*2 -> p = 0.
Proof.
  elim/lt_wf_ind: n p => n.                                    (* 整礎帰納法 *)
  case: (posnP n) => [-> _ [] // | Hn IH p Hnp].
Admitted.

(* 無理数 *)
Require Import Reals Field.              (* 実数とそのための field タクティク *)

Definition irrational (x : R) : Prop :=
   forall (p q : nat), q <> 0 -> x <> (INR p / INR q)%R.

Theorem irrational_sqrt_2: irrational (sqrt (INR 2)).
Proof.
  move=> p q Hq Hrt.
  apply /Hq  /(main_thm p) /INR_eq.
  rewrite -mul2n !mult_INR -(sqrt_def (INR 2)) ?Hrt; last by auto with real.
  have Hqr : INR q <> 0%R by auto with real.
  by field.
Qed.
```

練習問題 3.1 Admitted を Qed に変え, 証明を完成せよ.