続 総和

Jacques Garrigue, 2019 年 7 月 10 日

1 前回の課題

```
Lemma odd_square n : odd n = odd (n*n).
Proof. by rewrite odd_mul andbb. Qed.
Lemma even_double_half n : ~~odd n -> n./2.*2 = n.
Proof. by rewrite -[RHS]odd_double_half => /negbTE ->. Qed.

(* 本定理 *)
Theorem main_thm (n p : nat) : n * n = (p * p).*2 -> p = 0.
Proof.
  elim/lt_wf_ind: n p => n.
  case: (posnP n) => [-> _ [] // | Hn IH p Hnp].
  have Hn2 : (n./2 < n)%coq_nat.
    apply/ltP; by rewrite -divn2 ltn_Pdiv.
  have even_n : ~~odd n by rewrite odd_square Hnp odd_double.
  move: Hnp; rewrite -(even_double_half n) //.
  rewrite -muln2 mulnAC mulnA !muln2. move /double_inj => Hnp.
  have even_p : ~~odd p by rewrite odd_square -Hnp odd_double.
  move: Hnp; rewrite -(even_double_half p) // => /esym.
  rewrite -muln2 mulnAC mulnA !muln2. move /double_inj /esym /IH => -> //.
Qed.
```

2 環と ring タクティク

Coq には ring という便利なタクティクがある．演算子の環の構造を使い，方程式を自動的に解く．

```
Module Ring.
Require Import ZArith.
Open Scope Z_scope.

Variables a b c : Z.

Goal a + b + c = b + c + a.
Proof. ring. Qed.

Goal a * (b + c) = c * a + a * b.
Proof. ring. Qed.

Goal a - b + c = a + c - b.
Proof. ring. Qed.
End Ring.
```

ring は自動化のタクティクではあるものの，mathcomp でも使える．ただし，自然数は環ではなく半環なので，自動的に解ける方程式が制限される．omega は仮定にある不等式も使えてより強いが，Coq の定義に戻す必要がある．

```
Module SemiRing.
Variables a b c : nat.

Goal a + b + c = b + c + a.
Proof.
  ring.
  Show Proof.
Qed.

Goal a * (b + c) = c * a + a * b.
Proof. ring. Qed.

Require Import Omega.
Goal a >= b -> a - b + c = a + c - b.
Proof.
  move=> ab.
  ring.
  rewrite addnC addnBA // addnC //.
Restart. (* Coq の定義に戻して omega *)
  rewrite -!(plusE,minusE,multE) => /leP ab.
  omega.
Qed.
End SemiRing.
```

そのそも，ring と omega, field と fourier でできるのは基本演算子だけなので，冪などは手で証明せざるを得ない．

今回は基本的な補題に加えて，ssrnat の以下の補題を使えば証明できる．

```
subn1     : n - 1 = n.-1
addKn     : m + n - m = n
subnDA    : n - (m + p) = n - m - p
addnBA    : p <= n -> m + (n - p) = m + n - p
subnK     : m <= n -> n - m + m = n
prednK    : 0 < n -> n.-1.+1 = n
exp1n     : 1 ^ n = 1
expn0     : m ^ 0 = 1
expn1     : m ^ 1 = m
expn_gt0  : (0 < m ^ n) = (0 < m) || (n == 0)
ltn_exp2r : 0 < e -> (m ^ e < n ^ e) = (m < n)
leq_pexp2l : 0 < m -> n1 <= n2 -> m ^ n1 <= m ^ n2
```