

数学：総和・二項係数

Jacques Garrigue, 2021 年 12 月 8 日

1 有限型：fintype

MathComp で有限個の元しか含まない型が特別な役割を果たしている。例えば、定義域が有限型の関数は n 組として見る事ができるので、値域が `eqType` に属していれば、関数同士も比較できる。

`eqType` と同様に、有限型のソート `finType` が提供されている。 `eqType` と同様に、 `finType` の範疇が直積や直和で閉じられている。

基本的な有限型として、 `bool` 以外に各 $n : \text{nat}$ に対して n より小さい自然数の型 `'I.n` も定義されている。

また、各 $T : \text{finType}$ に対し、 `#|T|` は T の元の数、 `enum T` は T の元のリストである。

```
Definition enum (T : finType) : seq T := ... (* T の全ての元の列 *)
```

```
Lemma cardE (T : finType) : #|T| = size (enum T).
```

```
Check ord0 : 'I_1.
```

```
Goal #|'I_3| = 3.
```

```
Proof. by rewrite card_ord. Qed.
```

```
(* val が 'I_n の値を nat に戻す *)
```

```
Goal [seq val i | i <- enum 'I_3] = [:: 0; 1; 2].
```

```
Proof. by rewrite enumT /= unlock /= val_ord_enum. Qed.
```

2 総和：bigop

MathComp の `bigop` モジュール (ファイル `ssreflect/bigop.v`) が総和や総乗を定義している。基本的な定義は

```
Variables (R : Type) (op : R -> R -> R) (un : R).
```

```
Variables (I : Type) (s : seq I) (P : I -> bool) (F : I -> R).
```

```
Definition \big[op/un]_(i <- s | P i) F i :=
```

```
  foldr op un [seq F i | i <- s & P i].
```

```
  (* P(i) をみたま s の要素に F を掛け、op でまとめる *)
```

例えば、`\sum` は `\big[addn,0]`、`\prod` は `\big[muln,1]` の記法である。

最も基本的な性質は合同によって証明される。

```
Lemma eq_bigr F1 F2 : (forall i, P i -> F1 i = F2 i) ->
```

```
  \big[op/un]_(i <- r | P i) F1 i = \big[op/un]_(i <- r | P i) F2 i.
```

`\big[* , e]` を意味のある演算に使うために $(*, e)$ は最低でもモノイドでなければならない。結合律をみたし、任意の $r : R$ について、 $r * e = r = e * r$ 。補題によって、可換律も求めたり、2つの演算子の分配律を求めることもある。 `eqType` と同様に、登録は `Canonical Structure` を使う。

```
Canonical addn_monoid := Law addnA addn0n addn0. (* + はモノイド *)
```

```
Canonical addn_comoid := ComLaw addnC. (* + は可換 *)
```

```
Canonical addn_addoid := AddLaw mulnDl mulnDr. (* + と * の分配律 *)
```

```

Canonical muln_monoid := Law mulnA mul1n muln1.          (* * はモノイド *)
Canonical muln_muloid := MulLaw mul0n muln0.            (* 0 が * の吸収元 *)
...
Canonical andb_monoid := Law andbA andTb andbT.        (* && はモノイド *)
Canonical orb_monoid := Law orbA orFb orbF.            (* || はモノイド *)
Canonical maxn_monoid := Law maxnA max0n maxn0.        (* maxn はモノイド *)
Canonical cat_monoid T := Law (@catA T) (@cat0s T) (@cats0 T). (* ++ *)
...

```

以上の構造を使って、様々な演算子`\prod`, `\max`, `\bigcup`, `\bigcap` が定義される。

具体的な演算子に関する補題の例

```

Lemma big1 I r (P : pred I) F :                          (* [op,un] がモノイドの場合 *)
  (forall i, P i -> F i = un) -> \big[op/un]_(i <- r | P i) F i = un.
Lemma big_split I r (P : pred I) F1 F2 :                 (* [op/un] が可換モノイドの場合 *)
  \big[op/un]_(i <- r | P i) op (F1 i) (F2 i) =
  op (\big[op/un]_(i <- r | P i) F1 i) (\big[op/un]_(i <- r | P i) F2 i).

```

さらに、範囲について様々な表記がある。まず、`P` を省略できる。

```

Definition \big[op/un]_(i <- s) F i := foldr op un [seq F i | i <- s].

```

自然数の範囲が使える。

```

Fixpoint iota m n := if n is n'.+1 then m :: iota m.+1 n' else [::].
Definition \big[op/un]_(m <= i < n | P i) F i :=
  \big[op/un]_(i <- iota m n | P i) F i. (* iota m n = [:: m;m+1;...;m+n-1] *)

```

また、有限型 (`finType`) を添えじとして使える。

```

Variable T : finType.
Definition \big[op/un]_(i : T | P i) F i :=
  \big[op/un]_(i <- enum T | P i) F i.

```

`n` より小さい自然数の型 `'I_n` も `finType` に属する。その場合、特別な表記が使える。

```

Variable n : nat
Definition \big[op/un]_(i < n | P i) F i := \big[op/un]_(i : 'I_n | P i) F i.

```

添字に関する補題がとても多い。数例だけを見せる。

```

Lemma big_cat_nat n m p (P : pred nat) F : m <= n -> n <= p ->
  \big[op/un]_(m <= i < p | P i) F i =
  op (\big[op/un]_(m <= i < n | P i) F i) (\big[op/un]_(n <= i < p | P i) F i).
Lemma big_nat1 n F : \big[op/un]_(n <= i < n.+1) F i = F n.
Lemma big_mkord :                                     (* 'I_n が nat に自動変換されるため *)
  \big[op/un]_(0 <= i < n | P i) F i = \big[op/un]_(i < n | P i) F i

```

3 二項係数

`mathcomp/ssreflect/binomial.v` で二項係数が定義されている。

主な定義と定理は以下のとおり。

```

Notation 'C(n,m) := (binominal n m)                    (* 二項係数  $_n C_m$  *)

```

```

Lemma bin0 n : 'C(n, 0) = 1.

```

```

Lemma bin0n m : 'C(0, m) = (m == 0).

```

```

Lemma binS n m : 'C(n.+1, m.+1) = 'C(n, m.+1) + 'C(n, m).
Lemma bin1 n : 'C(n, 1) = n.
Lemma binn n : 'C(n, n) = 1.
Lemma binSn n : 'C(n.+1, n) = n.+1.
Theorem Pascal a b n :
  (a + b) ^ n = \sum_(i < n.+1) 'C(n, i) * (a ^ (n - i) * b ^ i).

```

名古屋大学 2013 年度の入試問題

k, m, n は自然数で、 $k \geq 0, m \geq 2, n \geq 1$ とする。

$$S_k(n) = \sum_{i=1}^n i^k \qquad T_m(n) = \sum_{k=1}^{m-1} {}_m C_k S_k(n)$$

1. $T_m(1)$ と $T_m(2)$ を求めよ。
2. 一般的な n に対して、 $T_m(n)$ を求めよ。
3. p が 3 以上の素数のとき、 $S_k(p-1)$ ($1 \leq k \leq p-2$) は p の倍数であることを証明せよ。

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

```
Section Nagoya2013.
```

```
Definition Sk k n := \sum_(1 <= i < n.+1) i^k.
```

```
Variable m : nat.
```

```
Hypothesis Hm : m > 1.
```

```
Definition Tm n := \sum_(1 <= k < m) 'C(m,k) * Sk k n. (* binomial.v 参照 *)
```

```
Lemma Sk1 k : Sk k 1 = 1.
```

```
Proof. by rewrite /Sk big_nat1 exp1n. Qed.
```

```
Lemma Tm1 : Tm 1 = 2^m - 2.
```

```
Proof.
```

```
  rewrite /Tm.
```

```
  rewrite [in 2^m](_ : 2 = 1+1) //.
```

```
  rewrite Pascal.
```

(* 二項公式 *)

```
  transitivity ((\sum_(0 <= k < m.+1) 'C(m,k)) - 2).
```

```
    symmetry.
```

```
    rewrite (@big_cat_nat _ _ _ m) // =.
```

```
    rewrite (@big_cat_nat _ _ _ 1) // =; last by apply ltnW.
```

```
    rewrite addnAC !big_nat1 bin0 binn addKn.
```

```
    apply eq_bigr => i H.
```

```
    by rewrite Sk1 muln1.
```

```
  rewrite big_mkord.
```

```
  congr (_ - _).
```

```
  apply eq_bigr => i _.
```

```
  by rewrite !exp1n !muln1.
```

```
Qed.
```

```
Search (_ ^ _) "exp". (* 自然数の指数関数 expn に関する様々な補題 *)
```

Lemma Tm2 : Tm 2 = 3^m - 3.

Proof.

```
rewrite /Tm.
have ->: 3m - 3 = 2m - 2 + (3m - 1 - 2m).
  admit.
rewrite -Tm1.
rewrite [in 3m](_ : 3 = 1+2) //.
rewrite Pascal.
transitivity (Tm 1 + (\sum_(1 <= k < m) 'C(m,k) * 2k)).
  rewrite -big_split /=.
  apply eq_bigr => i _ .
  rewrite /Sk !big_cons !big_nil.
  by rewrite !addn0 -mulnDr.
congr (_ + _).
transitivity ((\sum_(0 <= k < m.+1) 'C(m,k) * 2k) - 1 - 2m).
```

Admitted.

Theorem Tmn n : Tm n.+1 = n.+2^m - n.+2.

Proof.

```
elim:n => [|n IHn] /=.
  by apply Tm1.
have Hm': m > 0 by apply ltnW.
have ->: n.+3m - n.+3 = n.+2m - n.+2 + (n.+3m - 1 - n.+2m).
```

Admitted.

Theorem Skp p k : p > 2 -> prime p -> 1 <= k < p.-1 -> p %| Sk k p.-1.

Admitted.

End Nagoya2013.

練習問題 3.1 上記の証明の admit と Admitted をなくせ。(Skp は著者も証明していない)

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
```