Coqゼミ　第5回

浅井研究室（担当：廣田知子）

2008年7月10日

1　帰納的なデータ型

データの定義をするときに、そのデータであるような条件を列挙することにより定義する方法。Inductiveコマンドを用いる。

最も簡単な例はbool型。

```
Coq < Print bool.
Inductive bool : Set :=
    true : bool
  | false : bool
```

bool型は、trueであるかfalseであるかのどちらかである、と定義している。これをOcamlで書けば、
`type bool = True | False` である。帰納的なデータ型はユーザが自由に定義できる。以下に例を示す：

```
Inductive season : Set :=
    Spring : season
  | Summer : season
  | Fall : season
  | Winter : season.
```

この定義をCoqに読み込ませれば、以下の様な表示がされる筈である。

```
season is defined
season_rect is defined
season_ind is defined
season_rec is defined
```

データ型seasonを定義したことにより、`season_rect, season_ind, season_rec` の三つの定理が自動的に定義された。次の節でそれぞれの定理についてみていく。

2　帰納的なデータ型に付随する定理

`season_ind`　帰納法の定理。

```
Coq < Check season_ind.
season_ind :
  forall P : season -> Prop,
  P Spring -> P Summer ->
  P Fall -> P Winter ->
  forall s : season, P s
```

つまり、四つの季節全てについてPが成り立つなら、任意の季節についてPが成り立つ。（再帰が無いので、単なる場合分けと同じ）

`season_rec`
　帰納法の定理のプログラムレベル版。

```
Coq < Check season_rec.
season_rec : forall P : season -> Set,
  P Spring -> P Summer ->
  P Fall -> P Winter ->
  forall s : season, P s
```

四つの季節の値を渡しておくと、季節を受け取ったら対応する値を返す関数になる。例えば、

```
Definition f :=
  season_rec (fun _ => nat) 0 1 2 3.
```

は、

```
Definition f (s : season) : nat :=
  match s with
    Spring => 0
  | Summer => 1
  | Fall => 2
  | Winter => 3
  end.
```

と同じ。

`season_rect`
　上二つの定理を定義しているおおもとだが、このゼミでは扱わない。

3　場合分け（case コマンド）

（再帰を含まない）帰納的なデータ型に対する証明は、場合分けとなる。例えば、season 型の s について場合分けしたいときには、

```
case s.
```

とする。（場合分けというのは、帰納法の特殊形である。このコマンドは season_ind を適用していると思えば良い。`induction s.` としても同じ。）

4　引数付きの帰納的なデータ型

例えば、年号に数字を引数として持つようなデータ型を定義したいとき、以下の様に書くことが出来る：

```
Inductive nengou : Set :=
    Meiji : nat -> nengou
  | Taisho : nat -> nengou
  | Showa : nat -> nengou
  | Heisei : nat -> nengou.
```

引数を持つ以外は season 型の定義と同じである。

```
Coq < Check nengou_ind.
nengou_ind :
  forall P : nengou -> Prop,
  (forall n : nat, P (Meiji n)) ->
  (forall n : nat, P (Taisho n)) ->
  (forall n : nat, P (Showa n)) ->
  (forall n : nat, P (Heisei n)) ->
  forall n : nengou, P n
Coq < nengou_rec.
nengou_rec :
  forall P : nengou -> Set,
  (forall n : nat, P (Meiji n)) ->
  (forall n : nat, P (Taisho n)) ->
  (forall n : nat, P (Showa n)) ->
  (forall n : nat, P (Heisei n)) ->
  forall n : nengou, P n
```

5　再帰を含む帰納的なデータ型

代表的なのは自然数。

```
Coq < Print nat.
Inductive nat : Set :=
    0 : nat
  | S : nat -> nat
```

S は引数付きである。

6　再帰を含む帰納的なデータ型に付随する定理

nat_ind　数学的帰納法そのもの。

```
Coq < Print nat_ind.
nat_ind =
fun P : nat -> Prop => nat_rect P
  : forall P : nat -> Prop,
    P 0 ->
    (forall n : nat, P n -> P (S n)) ->
    forall n : nat, P n
```

induction コマンドの引数に自然数を指定すると使われる。

nat_rec　自然数を受け取る関数を作るもととなるもの。

```
Coq < Print nat_rec.
nat_rec =
fun P : nat -> Set => nat_rect P
  : forall P : nat -> Set,
    P 0 ->
    (forall n : nat, P n -> P (S n)) ->
    forall n : nat, P n
```

0 の場合の返り値と、「n の場合の再帰の結果を受け取ったら、n+1 の場合の答えを返す関数」を受け取ったら、全ての n について答えを計算する関数を返す。これを使って自然数上の再帰関数を定義することが出来る。

7　再帰関数の定義

上の nat_rec を使っても自然数上の再帰関数を定義出来るが、普通は以下の方法を用いる。この例では、引数 n, m を受け取ったら、n+m を返す関数 plus を、再帰的に定義している。

```
Fixpoint
 plus (n m :nat) {struct n} :nat :=
 match n with
   0 => m
 | S p => S (plus p m)
 end.
```

struct は、どの変数についての再帰かを明示している。

この関数は勿論、nat_rec を使っても定義出来る。

```
Definition plus (m:nat) :=
  nat_rec
   (fun _ => nat) m
   (fun (n:nat) (x:nat) => S x).
```

8 帰納法（induction コマンド）

再帰を含む帰納的なデータ型（例えば x）に対して、帰納法を用いて証明するときは、induction x. を用いる。

以下に、自然数についての帰納法（数学的帰納法）を用いた証明の例を示す。

```
Theorem plus_assoc :
  forall x y z : nat,
  (x + y) + z = x + (y + z).
Proof.
  intros.
  induction x.
  simpl; reflexivity.
  simpl; rewrite IHx; reflexivity.
Qed.
```

simpl は現在の Goal に対して「適当に」簡単化を行ってくれるコマンド。

simpl (式). で、() で括られた式の部分だけを簡単化してくれる。

9 補足（tactic）

auto.

自明な Goal に対して自動的に証明を行ってくれる。（例えば、2 * 3 = 6）

trivial.

auto とほぼ同じ。

練習問題

問 1

1 月から *12* 月までを列挙する型 month を定義せよ。次に、月を受け取ったらその月の日数を返す関数 days を定義せよ。任意の月について、日数は *31* 以下であることを証明せよ。

月を受け取ったら、その次の月を返す関数 next_month を定義せよ。次の月が *8* 月なら現在は *7* 月であることを証明せよ。（仮定に H : January = August のようなものが出てきたら、この時点でこの等式は成り立たないので、無条件に *Goal* は成り立つ筈である。こういうときは discriminate H. とすれば良い。）

次の月が *3* 月なら、その月はどの月よりも日数が少ないことを証明せよ。

問 2

年号を受け取ったら、西暦年を返す関数を定義せよ。

問 3 次を証明せよ。

```
(1) Lemma plus_n_0 :
      forall n : nat, n = n + 0.
(2) Lemma plus_n_m :
      forall n m : nat, n + m = m + n.
```

途中で別の補題が必要になるかもしれない。その場合は、それをあらかじめ別途、定義・証明せよ。

問 4

自然数を受け取ったら、0 から受け取った自然数までの和を求める関数 sum_n を定義せよ。