Coqゼミ　第6回

浅井研究室（担当：廣田知子）

2008年7月17日

1　リスト（多相の帰納的なデータ型の例）

帰納的データ型は引数を持つことが出来る。この引数を用いて、多相の帰納的なデータ型を定義することが出来る。（考え方は多相恒等関数と同じ。）

一般的な例はリスト。Coqにあらかじめ定義されている。それを使えば、リストを`cons 3 nil`などと書ける。又は、
`3 :: nil`と書いても良い。

```
Coq < Require Import List.
Coq < Print list.
Inductive list (A : Type) : Type :=
    nil : list A
  | cons : A -> list A -> list A
For nil: Argument A is implicit
For cons: Argument A is implicit
```

最後の二行は、単に`cons 3 nil`と書けば、3がnatなので、そこから自動的にAがnatであると補われることを示している。

自分でこのような型を定義するときには、

```
Implicit Arguments nil [A].
Implicit Arguments cons [A].
```

とする。もしこの定義をしなければ、データの表記は、`cons nat 3 (nil nat)`のようになる。又、ペアもリストと同様に定義されている。

```
Coq < Print prod.
Inductive
  prod (A:Type) (B:Type) : Type :=
              pair : A -> B -> A * B
For pair: Arguments A, B are implicit
```

2　帰納的な命題の定義

帰納的なデータ型の定義と同じく、inductiveコマンドを用いる。

例えば、その値が偶数であることを表す命題は以下の様に書ける：

```
Inductive even : nat -> Prop :=
    O_even : even 0
  | plus_2_even : forall n:nat,
                  even n -> even (S (S n)).
```

これを定義すると、以下の様な表示がされる（`even_ind`は帰納法の定理）：

```
even is defined
even_ind is defined
```

この命題を使って定理・補題を定義することも出来る。例えば「4は偶数である」という補題は以下の様に定義出来る：

```
Lemma le_even_4 : even 4.
 Proof.
  apply plus_2_even. apply plus_2_even.
  apply O_even.
 Qed.
```

証明の際、`O_even`等の名前は、そのまま規則として使える。命題に対応する帰納法を使うときには`apply even.`とする。（又は`eapply even`。eがつくと、値がよく分からない変数について「そういう変数が存在して」という形のGoalを作ってくれる。）

又、仮定の中に帰納的な命題を満たしているものがある場合には、それをelimすると、帰納的定義のどれかが成り立っていた筈、というGoalを作ってくれる。

elimの代わりにinversionを使うと上手くいく場合もある。後者は、構成子が異なるという条件を加味して、どの場合が成り

立っていたのかを推論してくれる。(が、elim の機能を完全に含んでいる訳ではないらしい。)

3 色々な帰納的な命題

命題論理における、
「真 (True)」「偽 (False)」「かつ (and /\)」「または (or \/)」「存在して (ex)」等は、みな帰納的な命題として定義されている。

又、n <= m を示す命題 le n m は以下の様に定義されている:

```
Coq < Print le.
Inductive le (n:nat) : nat -> Prop :=
    le_n : n <= n
  | le_S : forall m:nat,
           n <= m -> n <= S m
```

4 補足 1 (tactic)

証明の際、
repeat (証明コマンド 1 ; 証明コマンド 2 ; ... ; 証明コマンド n).
とすると、(...) が適当なところまで繰り返される。

5 補足 2 (関数定義)

関数定義の際、条件文として if 文を使うことが出来る。書き方は Ocaml と一緒。
if 式 1 then 式 2 else 式 3

練習問題

問 1
要素の型が A 型であるような二分木の型である、btree A 型を定義せよ。この型を持つデータの名前は Empty と Node にせよ。

Node は引数として、左の木、自身が持つ要素 (A 型)、右の木を持つ。

問 2
リストがソートされていることを示す命題 sorted を定義せよ。

問 3
リスト (1 :: 3 :: 8 :: nil) がソートされていることを証明せよ。

問 4
任意の自然数 n と任意の自然数のリスト l について、n :: l がソートされていたら、l もソートされていることを示せ。

問 5
任意の自然数 n と既に昇順に並んでいる自然数のリスト l を受け取ったら、n を l の中の昇順に並ぶ位置に挿入したリストを返す関数 insert を定義せよ。

自然数 m と n について、m < n であるかどうかには、le_gt_dec m n を使え。(< は bool でなく Prop を返すので使えない。)

なお、le_gt_dec を使うには、
Require Import Compare_dec. とあらかじめ宣言しておくこと。

問 6
任意の自然数 n と任意の自然数のリスト l について、既に l が昇順に並んでいたら、insert n l も昇順に並んでいることを示せ。

ヒント: x > n から n <= x を示すには auto with arith. を使え。

問 7
任意の自然数 n と任意の自然数のリスト l について、l の中に n が何回出てくるかを返す関数 nb_occ を定義せよ。

二つの自然数が等しいかどうかには eq_nat_dec n n' を使え。(= は bool ではなく Prop を返すので使えない。)

なお、eq_nat_dec を使うには、
Require Import Peano_dec. とあらかじめ宣言しておくこと。