Jacques Garrigue, 2020 年 11 月 25 日

Mathcompでのプログラム証明

1 eqTypeとseq

Mathcomp ではある型 T における等価性が判定可能ということをソートみたいに表現できる．2 つの T の値が等価かどうかを計算する T -> T -> bool の関数があることを T : eqType と宣言できる．そうすれば，a と b の型が T なら，a == b がその関数を呼ぶ．

実は eqType は Type や Set のようなソートではなく，その関数とその正しさの証明を含むデータ構造である．

```
Record eqType : Type := Pack
  { T : Type; op : T -> T -> bool; _ : forall x y, reflect (x = y) (op x y) }
```

ただし，たとえば nat : eqType と書いても，本当は nat : Set なのでそのまま eqType の値ではないので，あらかじめ宣言された nat_eqType が自動的に使われ，区別する必要があまりない．

MathComp のリストの関数はモジュール seq で定義されている．因に，その中で Notation seq := list と宣言されているので，MathComp を使うと全てのリストの型が seq と表示される．

リストに関する関数は等価性の判定が必要なもの以外は普通に一般的な Type に対して定義されているが，各関数の性質を表した述語などがほとんどの場合 eqType を仮定している．特に，ある元 t : T : eqType がリスト l : seq T に含まれることを t \in l と書く．

また，ビュー機構を使い，bool 上の述語と量化子を使った表現を結びつけている．たとえば，全称は，

```
Variables (T : eqType) (a : T -> bool).
Fixpoint all s := if s is x :: s' then a x && all s' else true.
Lemma allP s : reflect (forall x : T, x \in s -> a x) (all a s).
```

2 整列の証明, 再び

```
Section sort.
Variable A : eqType.
Variable le : A -> A -> bool.
Variable le_trans: forall x y z, le x y -> le y z -> le x z.
Variable le_total: forall x y, ~~ le x y -> le y x.

Fixpoint insert a l := match l with
  | nil     => (a :: nil)
  | b :: l' => if le a b then a :: l else b :: insert a l'
  end.

Fixpoint isort l :=
  if l is a :: l' then insert a (isort l') else nil.

Fixpoint sorted l :=                    (* all を使って bool 上の述語を定義する *)
  if l is a :: l' then all (le a) l' && sorted l' else true.

Lemma le_seq_insert a b l :
  le a b -> all (le a) l -> all (le a) (insert b l).
```

```
Proof.
  elim: l => /= [-> // | c l IH].
  move=> leab /andP [leac leal].
  case: ifPn => lebc /=.
  - by rewrite leab leac.
  - by rewrite leac IH.
Qed.

Lemma le_seq_trans a b l :
  le a b -> all (le b) l -> all (le a) l.
Proof.
  move=> leab /allP lebl.
  apply/allP => x Hx.
  by apply/le_trans/lebl.
Qed.

Theorem insert_ok a l : sorted l -> sorted (insert a l). Admitted.
Theorem isort_ok l : sorted (isort l). Admitted.

(* perm_eq が seq で定義されているが補題だけを使う *)
Theorem insert_perm l a : perm_eq (a :: l) (insert a l).
Proof.
  elim: l => //= b l pal.
  case: ifPn => //= leab.
  by rewrite (perm_catCA [:: a] [:: b]) perm_cons.
Qed.

perm_trans : forall (T : eqType), transitive (seq T) perm_eq
Theorem isort_perm l : perm_eq l (isort l). Admitted.
End sort.

Check isort.
Definition isortn : seq nat -> seq nat := isort _ leq.
Definition sortedn := sorted _ leq.
Lemma leq_total a b : ~~ (a <= b) -> b <= a. Admitted.

Theorem isortn_ok l : sortedn (isortn l) && perm_eq l (isortn l). Admitted.

Require Import Extraction.
Extraction "isort.ml" isortn.                          (* コードが分かりにくい *)
```

OCaml で正しく動く.

```
% ocamlc -c isort.mli isort.ml
% ocaml
        OCaml version 4.07.1
# #load"isort.cmo";;
# open Isort;;
# isortn (Cons (S O, Cons (O, Cons (S (S O), Nil))));;
- : Isort.nat Isort.list = Cons (O, Cons (S O, Cons (S (S O), Nil)))
```

練習問題 2.1 Admitted を Proof に変え，証明を完成させよ.

3 偶奇，mathcomp 式

Prop での述語を使ったとき，述語の導出に対する帰納法の重要性を見た．しかし，mathcomp 式に述語を bool で定義すると，通常の帰納法で足りることが多い．

odd は ssrnat で bool の述語とした定義されている．even をその否定として定義すると様々なことが簡単になる．

```
Section even_odd.
Notation even n := (~~ odd n).              (* 単なる表記なので，展開が要らない *)

Theorem even_double n : even (n + n).
Proof. elim: n => // n. by rewrite addnS /= negbK. Qed.

(* 等式を使って n に対する通常の帰納法を可能にする *)
Theorem even_plus m n : even m -> even n = even (m + n).
Proof.
  elim: n => /= [|n IH] Hm.
  - by rewrite addn0.
  - by rewrite addnS IH.
Qed.

Theorem one_not_even : ~~ even 1.
Proof. reflexivity. Qed.

Theorem even_not_odd n : even n -> ~~ odd n.
Proof. done. Qed.

Theorem even_odd n : even n -> odd n.+1. Admitted.
Theorem odd_even n : odd n -> even n.+1. Admitted.
Theorem even_or_odd n : even n || odd n. Admitted.
Theorem odd_odd_even m n : odd m -> odd n = even (m+n). Admitted.
End even_odd.
```

練習問題 3.1 Admitted を Proof に変え，証明を完成させよ．