Jacques Garrigue, 2019 年 6 月 19 日

命題論理の証明論

1 証明論

第 2 回目の授業で見た証明論を前回の体系に合わせて少し修正する．Δ は命題の有限集合とする．

公理　$\Delta \vdash P \quad (P \in \Delta)$

$\supset$ 導入	$\dfrac{\Delta, P \vdash Q}{\Delta \vdash P \supset Q}$	$\wedge$ 導入	$\dfrac{\Delta \vdash P \quad \Delta \vdash Q}{\Delta \vdash P \wedge Q}$
$\supset$ 除去	$\dfrac{\Delta \vdash P \quad \Delta \vdash P \supset Q}{\Delta \vdash Q}$	$\wedge$ 除去	$\dfrac{\Delta \vdash P \wedge Q}{\Delta \vdash P} \quad \dfrac{\Delta \vdash P \wedge Q}{\Delta \vdash Q}$
$\neg$ 導入	$\dfrac{\Delta, P \vdash Q \quad \Delta, P \vdash \neg Q}{\Delta \vdash \neg P}$	$\vee$ 導入	$\dfrac{\Delta \vdash P}{\Delta \vdash P \vee Q} \quad \dfrac{\Delta \vdash Q}{\Delta \vdash P \vee Q}$
$\neg$ 除去	$\dfrac{\Delta \vdash \neg\neg P}{\Delta \vdash P}$	$\vee$ 除去	$\dfrac{\Delta \vdash P \vee Q \quad \Delta, P \vdash R \quad \Delta, Q \vdash R}{\Delta \vdash R}$

証明論にとって重要な定理が二つある．

定理 1 (健全性) 空の仮定の元で P が証明できるなら，意味論において P は恒真式である．形式的には，$\emptyset \vdash P$ が導出できれば $\forall v, [\![P]\!]_v = \mathit{true}$.

定理 2 (完全性) 意味論において P は恒真式ならば，空の仮定の元で P が証明できる．形式的には，$\forall v, [\![P]\!]_v = \mathit{true}$ ならば $\emptyset \vdash P$ が導出できる．

健全性の証明は比較的に簡単だが，完全性は論理積標準形に移る必要があり，かなり長い．

2 帰納的述語と導出

前節の導出規則から作った証明木は Coq では自然と帰納的述語として表現される．帰納的述語には 2 つの役割がある．

- 導出木の構造の表現．各節でどの規則がどう使われたかが分かる．
- 導出規則が正しく使われていることの保証．型付けがそれを確認する．

`bool` 上の述語関数では後者が表現できても，前者は表現できない．

両方を併わせると，帰納法によって導出木を分解することができ，通常だと逆転補題によて得られる前提条件が同時に得られる．

3 実装

```
From mathcomp Require Import all_ssreflect.
Set Implicit Arguments.

(* 命題の定義 *)
Inductive prop : Set :=
  | Atom of nat
```

```
  | Neg of prop
  | Conj of prop & prop
  | Disj of prop & prop.

Definition Imp p1 p2 := Disj (Neg p1) p2.
Notation tt := (Disj (Atom 0) (Neg (Atom 0))).
Notation ff := (Conj (Atom 0) (Neg (Atom 0))).

(* ... *)
Definition eval_lit (v : valuation) (l : lit) :=
  if l.2 then v l.1 else ~~ v l.1.                    (* 元の定義に戻す *)
(* ... *)

Lemma eval_neg_lit v a : eval_lit v (neg_lit a) = ~~ eval_lit v a.
Proof. case: a => a [] //=; by rewrite negbK. Qed.

Lemma eval_cnf_cat v l1 l2 :
 eval_cnf v (l1 ++ l2) = eval_cnf v l1 && eval_cnf v l2.
Proof. by rewrite /eval_cnf all_cat. Qed.

Lemma disj_correct v l1 l2 :
 eval_cnf v (disj l1 l2) = eval_cnf v l1 || eval_cnf v l2.
Proof.
  rewrite /disj.
  elim: l1 => //= a l1 IH.
  rewrite eval_cnf_cat IH orb_andl.
  congr (_ && _).                                     (* 合同を使う *)
  elim: l2 {IH l1} => /= [|b l2 IH].
    by rewrite orbT.
  by rewrite IH orb_andr eval_clause_cat.
Qed.

(* 論理積標準形は意味を変えない *)
Theorem cnf_correct v p : is_nnf p -> eval_cnf v (cnf p) = eval v p.
Proof.
  elim: p => [n | p IH | p1 IH1 p2 IH2 | p1 IH1 p2 IH2] //=.
  - by rewrite orbF andbT.
  - destruct p => //; by rewrite -IH //= !andbT !orbF.
  - move/andP => [] /IH1 <- /IH2 <-. by rewrite eval_cnf_cat.
  - move/andP => [] /IH1 <- /IH2 <-. by rewrite disj_correct.
Qed.

(* 補題 *)
Lemma eval_clause_true v (a : lit) (c : clause) :
  a \in c -> eval_lit v a -> eval_clause v c.
Proof.
  elim: c => //= b c IH.
  rewrite inE => /orP [].
Admitted.

Lemma tautology_notin a (c : seq lit) :
  neg_lit a \notin c ->
  (forall v, eval_clause v (a :: c)) -> (forall v, eval_clause v c).
Admitted.                                             (* 省略 *)
```

```
(* 恒真判定の正しさ *)
Lemma tauto_clause_ok c : reflect (forall v, eval_clause v c) (tauto_clause c).
Proof.
  elim: c => //= [|a c IH].
    by constructor => /(_ (fun => true)).
  case /boolP: (_ \in _) => //= Hna.
    constructor => v.                                    (* reflect ... true *)
    case Ha: eval_lit => //=.
    move/eval_clause_true: Hna; apply.
    by rewrite eval_neg_lit Ha.
  case: IH => Hv.
Admitted.

Lemma tauto_cnf_ok l : reflect (forall v, eval_cnf v l) (tauto_cnf l).
Proof.
  apply: (iffP allP) => /= [H v | H c Hc].
    apply/allP => /= c /H.
Admitted.

(* tauto_cnf は恒真式を正しく判定している *)
Theorem tauto_ok p : reflect (tautology p) (tauto_cnf (cnf (nnf false p))).
Proof.
  rewrite /tautology.
  apply: iffP.
Admitted.

(* 証明論 *)

(* 証明の判定 *)
Reserved Notation "h |- p" (at level 69).
Definition hypotheses := seq prop.

(* 証明の定義 *)
Unset Implicit Arguments.                                (* 引数を省略しない *)
Inductive provable : hypotheses -> prop -> Prop :=
  | AxiomI : forall (h : hypotheses) (p : prop), p \in h -> h |- p
  | ImpI   : forall h p1 p2, p1 :: h |- p2 -> h |- (Imp p1 p2)
  | ImpE   : forall p1 h p2, h |- p1 -> h |- Imp p1 p2 -> h |- p2
  | ConjI  : forall h p1 p2, h |- p1 -> h |- p2 -> h |- Conj p1 p2
  | ConjE1 : forall p2 h p1, h |- Conj p1 p2 -> h |- p1
  | ConjE2 : forall p1 h p2, h |- Conj p1 p2 -> h |- p2
  | DisjI1 : forall h p1 p2, h |- p1 -> h |- Disj p1 p2
  | DisjI2 : forall h p1 p2, h |- p2 -> h |- Disj p1 p2
  | DisjE  : forall p1 p2 h q,             (* 結論にない引数を最初に受け取る *)
               h |- Disj p1 p2 -> p1 :: h |- q -> p2 :: h |- q -> h |- q
  | NegI   : forall q h p, p :: h |- q -> p :: h |- Neg q -> h |- Neg p
  | NegE   : forall h p,   h |- Neg (Neg p) -> h |- p
where "h |- p" := (provable h p).                  (* 先に定義した記法を使う *)

(* 仮定を満たす割り当て *)
Definition validates v (h : hypotheses) := {in h, forall p, eval v p}.
(* 論理的帰結 *)
Definition consequence h p := forall v, validates v h -> eval v p.
(* validates のための補題 *)
Lemma in_consP (A : eqType) (P : A -> Prop) a s :
```

```
  P a -> {in s, forall x, P x} -> {in a :: s, forall x, P x}.
Proof. move=> Ha Hs x; by rewrite inE => /orP [/eqP -> | /Hs]. Qed.

(* 証明論の健全性 *)
Theorem soundness p h : h |- p -> consequence h p.
Proof.
  induction 1 => v Hv /=.                              (* 導出に対する帰納法 *)
  - by apply Hv.
  - case Hp1: (eval v p1) => //=.
    apply IHprovable.
    by apply in_consP.
  - move/IHprovable2: (Hv) => /=.                          (* Hv を残す *)
    by rewrite IHprovable1.
Admitted.

(* provable を auto のヒントにする *)
Hint Constructors provable.
(* AxiomI の特別な場合 *)
Lemma axiom_head p h : p :: h |- p.
Proof. by apply /AxiomI /mem_head. Qed.
Hint Resolve axiom_head.

(* 前提を強くしても証明が可能 *)
Theorem weakening h1 h2 h3 p :
  h1 ++ h2 |- p -> h1 ++ h3 ++ h2 |- p.
Proof.
  move eqh: (h1++h2) => h H.
  move: h1 h2 eqh.
  induction H => h1 h2 eqh; subst h; auto.
  - apply AxiomI.
    move: H; rewrite !mem_cat.
    case/orP => -> //.
    by rewrite !orbT.
  - apply ImpI.
    by apply (IHprovable (p1::h1)).                    (* 単一化のヒント *)
  - apply /ImpE /IHprovable2; auto.                       (* 連続 apply *)
Admitted.

(* よく使う特殊な場合 *)
Corollary weakening_head p h p' : h |- p' -> p::h |- p'.
Proof. apply (weakening nil h (p::nil)). Qed.
Hint Resolve weakening_head.

Lemma excluded_middle h p : h |- Disj p (Neg p).
Proof.
  apply NegE, (NegI p). (* 連続 apply, /lem でもいい *)
    apply NegE, (NegI (Disj p (Neg p))); auto.
  apply (NegI (Disj p (Neg p))); auto.
  Show Proof.
Qed.

Lemma ex_falso p q h : h |- Conj p (Neg p) -> h |- q.
Admitted.
```

練習問題 3.1 証明の中の Admitted を Qed に変えよ.