

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

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 $\forall v, \llbracket P \rrbracket_v = true$ 。

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、 $\forall v, \llbracket P \rrbracket_v = true$ ならば $\emptyset \vdash P$ が導出できる。

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

2 便利なコマンド

「case_eq 式」 destruct と同じだが、等式を仮定に置く。remember 式 as x; destruct x と同じ効果がある。

「apply <-」 同値関係を逆方向に使う。-> もある。

「specialize (仮定 引数)」 仮定を特定の引数に限定する。

「intuition」 auto より強い。論理和の場合わけも行う。ゴールを増やす可能性がある。

「try 作戦」 引数の作戦を実行し、失敗したら何もしない。

「Notation」 前回で見たように、演算子などを定義できる。unfold が要らない定数も定義できる。

「Reserved Notation」 記法を予約する。where と組み合わせて、Inductive の定義の中で Notation が再帰的に使える。

「unfold 定数」 定数の定義を展開する。

「remember 式 as」 式を変数に置き換え、仮定に等式を置く。induction と組み合わせて効果的に使える。

「clear 仮定」 不要な仮定を忘れる。「clear -仮定」で必要な仮定だけを残す。

「Hint Extern」 ゴールの形によって、自動的に作戦を実行する。

「solve [作戦]」 作戦を実行してゴールが解ければ成功。ゴールが残れば失敗する。

「revert 仮定」 仮定をゴールに戻す。帰納法を行うまでに使う。

3 実装

今日の実装は http://www.math.nagoya-u.ac.jp/~garrigue/lecture/2013_SS/ においてある。(先週の解答を含めて)

```
(* Clause の恒真判定の正しさ *)
Lemma eval_neg_lit : forall v a, eval_lit v (neg_lit a) = negb (eval_lit v a).
Lemma eval_clause_true : forall v a c,
  In a c -> eval_lit v a = true -> eval_clause v c = true.
Definition neg_val v a : {v' | eval_lit v' a = false /\
  forall b, b <> atom_lit a -> v' b = v b}.
Lemma tauto_clause_ok :
  forall c, tauto_clause c = true <-> (forall v, eval_clause v c = true).
Proof.
  induction c; simpl.
  split; intros; auto.
  apply H.
  exact (fun x => true).
  destruct lit_eq_dec.
  destruct a; discriminate.
  destruct In_dec.
  split; intros; auto.
  case_eq (eval_lit v a); intro Ha.
  auto.
  apply (eval_clause_true _ _ _ i).
  rewrite eval_neg_lit, Ha.
  reflexivity.
  split; intros.
  destruct (eval_lit v a); simpl; auto.
  apply -> IHc; auto.
  apply <- IHc; intros.
  destruct (neg_val v a) as [v' [v'_a v'_other]].
  specialize (H v').
  rewrite v'_a in H; simpl in H.
  clear IHc n.
  (* eval_clause v' c = true -> eval_clause v c = true を帰納法で証明 *)
  induction c; simpl in *; auto.
  case_eq (eval_lit v a0); simpl; intro Ha0.
  auto.
  apply IHc.
  intuition.
  destruct (atom_eq_dec (atom_lit a0) (atom_lit a)).
  destruct a0; destruct a; simpl in *; try discriminate; subst;
  try rewrite v'_a in *; auto.
  destruct a0; simpl in *; rewrite v'_other, Ha0 in H; auto.
Qed.
```

(* 意味論 *)

Definition hypotheses := list prop.

Notation tt := (Disj (Atom 0) (Neg (Atom 0))). (* 透明な定義 *)

Notation ff := (Conj (Atom 0) (Neg (Atom 0))).

(* 証明の判定 *)

Reserved Notation "h |- p" (at level 69).

(* 証明の定義 *)

```
Inductive provable : hypotheses -> prop -> Prop :=
| AxiomI : forall h p, In p 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,
  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 *)

```

Definition validates v h := forall p, In p h -> eval v p = true.
Definition consequence h p :=
  forall v, validates v h -> eval v p = true.

```

(* 論理的帰結 *)

(* 証明論の健全性 *)

```

Theorem soundness : forall p h, h |- p -> consequence h p.
Proof.
  unfold consequence.
  unfold validates.
  intros.
  induction H.
Admitted.

```

(* 完全性も証明する *)

(* provable を auto のヒントにする *)

```

Hint Constructors provable.
(* 特殊なヒント：ゴールが以下の形の時に、右の作戦で解こうとする *)
Hint Extern 2 ( _ :: _ |- _ ) => try solve [apply AxiomI; simpl; auto].
Goal forall h p, p :: h |- p.
  intros. apply AxiomI. simpl. Abort.

```

(* simpl すると自明になる *)

(* 前提を強くしても証明が可能 *)

```

Theorem weakening : forall h1 h2 h3 p,
  h1++h2 |- p -> h1++h3++h2 |- p.
Proof.
  intros.
  remember (h1++h2) as h.
  revert h1 h2 Heqh.
  induction H; intros; subst; auto.
  (* AxiomI *)
  apply AxiomI.
  apply in_or_app.
  destruct (in_app_or _ _ _ H); auto.
  right.
  apply in_or_app; auto.
Admitted.

```

(* よく使う特殊な場合 *)

```

Corollary weakening_head : forall p h p', h |- p' -> p::h |- p'.
  intros; apply (weakening nil h (p::nil) _ H).
Qed.
Hint Resolve weakening_head.

```

(* nnf への変換が証明可能性を保存する *)

```

Lemma provable_nnf : forall p h, h |- p <-> h |- nnf p.
Proof.
  intro.
  functional induction (nnf p); simpl; split; intros; auto;
  try apply IHp0; try apply <- IHp0 in H.
Admitted.

```

(* Clause に関する完全性 *)

```

Parameter true_clause : forall h,
  tauto_clause h = true -> exists b, In (Posi b) h / In (Nega b) h.

```

(* 論理積標準形を命題に戻す *)

```

Definition prop_of_lit (l : lit) : prop := Atom 0.
Definition prop_of_clause :=
  fold_right (fun x => Disj (prop_of_lit x)) ff.
Definition prop_of_cnf :=

```

(* 定義せよ *)

```

fold_right (fun x => Conj (prop_of_clause x)) tt.
Parameter prop_of_cnf_ok : forall v c, eval_cnf v c = eval v (prop_of_cnf c).

```

(* 論理積標準形の完全性 *)

```

Parameter provable_clause : forall a c h,
  In a c -> In (prop_of_lit a) h -> h |- prop_of_clause c.
Parameter completeness_cnf : forall p,
  (forall v, eval_cnf v p = true) -> nil |- prop_of_cnf p.

```

(* 補題 *)

```

Parameter excluded_middle : forall h p, h |- Disj p (Neg p).
Hint Resolve excluded_middle.
Parameter ex_falso : forall p q h, h |- Conj p (Neg p) -> h |- q.
Parameter prop_of_clause_weaken1 : forall b a h,
  h |- prop_of_clause a -> h |- prop_of_clause (b ++ a).
Parameter prop_of_clause_weaken2 : forall b a h,
  h |- prop_of_clause a -> h |- prop_of_clause (a ++ b).
Parameter provable_cnf_app : forall c1 c2 h,
  h |- prop_of_cnf c1 ->
  h |- prop_of_cnf c2 -> h |- prop_of_cnf (c1 ++ c2).
Parameter provable_cnf_app_inv : forall c1 c2 h,
  h |- prop_of_cnf (c1 ++ c2) ->
  h |- prop_of_cnf c1 /\ h |- prop_of_cnf c2.
Parameter prop_of_clause_app_inv : forall c1 c2 h,
  h |- prop_of_clause (c1 ++ c2) ->
  h |- Disj (prop_of_clause c1) (prop_of_clause c2).

```

```

Parameter provable_cnf_disj_l : forall c1 c2 h,
  h |- prop_of_cnf c1 -> h |- prop_of_cnf (disj c1 c2).

```

```

Proof.
  intros.
  induction c1; simpl in *. auto.
  specialize (IHc1 (ConjE2 _ _ _ H)).
  remember (disj c1 c2) as c3.
  clear Heqc3.
  induction c2; simpl in *.
  auto.
  apply ConjI.
  apply prop_of_clause_weaken2.
  apply (ConjE1 _ _ _ H).
  auto.

```

Qed.

```

Parameter provable_cnf_disj_r : forall c1 c2 h,
  h |- prop_of_cnf c2 -> h |- prop_of_cnf (disj c1 c2).
Parameter provable_disj_conj : forall a b c h,
  h |- Conj (Disj a c) (Disj b c) <-> h |- Disj (Conj a b) c.
Parameter provable_cnf_disj : forall c1 c2 h,
  h |- prop_of_cnf (disj c1 c2) ->
  h |- Disj (prop_of_cnf c1) (prop_of_cnf c2).

```

(* 論理積標準形への変換が証明可能性を保存する *)

```

Lemma provable_cnf : forall h p,
  is_nnf p -> (h |- p <-> h |- prop_of_cnf (cnf p)).

```

```

Proof.
  intros h p Hnnf. revert h.
  induction Hnnf; simpl; split; intros.
Admitted.

```

```

Theorem completeness : forall p, tautology p -> nil |- p.

```

```

Proof.
  intros.
  apply <- provable_nnf.
  apply <- provable_cnf; auto.
  apply completeness_cnf.
  intros.
  rewrite cnf_correct, nnf_correct; auto.

```

Qed.