

述語論理と帰納法

1 述語論理

前回見た命題論理は、推論の概念を捉えているが、具体的な対象に対して議論することができない。我々が一般的に使う論理はその拡張である述語論理になる。

論理式 論理式は以下の結合子から定義される。

t	::=	x	項変数
		c	項定数
		$f(t_1, \dots, t_n)$	項関数
P, Q	::=	\dots	命題
		$p(t_1, \dots, t_n)$	述語
		$\forall x.P$	全称
		$\exists x.P$	存在
		$t_1 = t_2$	等価性

導出規則 命題論理の導出規則に以下の規則を加える。

\forall 導入	$\frac{\Delta \vdash P \quad x \notin fv(\Delta)}{\Delta \vdash \forall x.P}$	\exists 導入	$\frac{\Delta \vdash [t/x]P}{\Delta \vdash \exists x.P}$
\forall 除去	$\frac{\Delta \vdash \forall x.P}{\Delta \vdash [t/x]P}$	\exists 除去	$\frac{\Delta \vdash \exists x.P \quad \Delta, P \vdash Q \quad x \notin fv(\Delta, Q)}{\Delta \vdash Q}$
反射率	$\Delta \vdash t = t$	代入	$\frac{\Delta \vdash [t_1/x]P \quad \Delta \vdash t_1 = t_2}{\Delta \vdash [t_2/x]P}$

ここで自由変数 $fv(P)$ と代入が利用される。

$$\begin{aligned}
 fv(x) &= \{x\} & fv(c) &= \emptyset & fv(f(t_1, \dots, t_n)) &= \bigcup fv(t_i) \\
 fv(True) &= fv(False) = fv(X) = \emptyset & & & fv(p(t_1, \dots, t_n)) &= \bigcup fv(t_i) \\
 fv(P \supset Q) &= fv(P \wedge Q) = fv(P \vee Q) = fv(P) \cup fv(Q) & & & & \\
 fv(\forall x.P) &= fv(\exists x.P) = fv(P) \setminus \{x\} & & & &
 \end{aligned}$$

$$\begin{aligned}
 [t/x]x &= t & [t/x]y &= y \\
 [t/x]c &= c & [t/x]f(t_1, \dots, t_n) &= f([t/x]t_1, \dots, [t/x]t_n)
 \end{aligned}$$

$$\begin{aligned}
 [t/x]True &= True & [t/x]False &= False & [t/x]X &= X \\
 [t/x](P \supset Q) &= [t/x]P \supset [t/x]Q & & & \wedge, \vee \text{も同様} & \\
 [t/x]p(t_1, \dots, t_n) &= p([t/x]t_1, \dots, [t/x]t_n) & & & & \\
 [t/x]\forall y.P &= \forall y.[t/x]P \quad (y \notin fv(t)) & & & [t/x]\forall x.P &= \forall x.P \\
 [t/x]\exists y.P &= \exists y.[t/x]P \quad (y \notin fv(t)) & & & [t/x]\exists x.P &= \exists x.P
 \end{aligned}$$

導出の例

$$\frac{\frac{\forall x. human(x) \supset mortal(x) \vdash \forall x. human(x) \supset mortal(x)}{\forall x. human(x) \supset mortal(x) \vdash human(S) \supset mortal(S)} \quad human(S) \vdash human(S)}{\forall x. human(x) \supset mortal(x), human(S) \vdash mortal(S)}$$

Coq との対応 Coq では項は Set に属する型、命題は Prop にそれぞれ入るが、扱いは同じである。

抽象と適用はあらゆる種類のものに対してできる。

$$\begin{array}{l} \text{抽象} \quad \frac{\Gamma, X : S \vdash M : T \quad \Gamma \vdash \forall X : S, T : \text{Type}}{\Gamma \vdash \text{fun } X : S \Rightarrow M : \forall X : S, T} \qquad \text{適用} \quad \frac{\Gamma \vdash M : \forall X : S, T \quad \Gamma \vdash A : S}{\Gamma \vdash M A : [A/X]T} \\ \\ \Gamma \vdash \text{Set} : \text{Type} \quad \frac{\Gamma \vdash A : \text{Set}}{\Gamma \vdash A : \text{Type}} \quad \frac{\Gamma \vdash A : S \quad \Gamma, X : A \vdash B : \text{Set}}{\Gamma \vdash \forall X : A, B : \text{Set}} \quad S \in \{\text{Set}, \text{Prop}\} \\ \\ \Gamma \vdash \text{Prop} : \text{Type} \quad \frac{\Gamma \vdash A : \text{Prop}}{\Gamma \vdash A : \text{Type}} \quad \frac{\Gamma \vdash A : S \quad \Gamma, X : A \vdash B : \text{Prop}}{\Gamma \vdash \forall X : A, B : \text{Prop}} \quad \vee \begin{array}{l} S \in \{\text{Set}, \text{Prop}\} \\ A \in \{\text{Set}, \text{Prop}\} \end{array} \end{array}$$

この抽象と適用は普通の関数の抽象と適用の代わりに使える。そもそも $A \rightarrow B$ は $\forall X : A, B$ の略記法でしかない。

上の導出の例は適用 2 回でできる。

Section Socrates.

```
Variable A : Set.
Variables human mortal : A -> Prop.
Variable socrates : A.
```

```
Hypothesis hm : forall x, human x -> mortal x.
Hypothesis hs : human socrates.
```

```
Theorem ms : mortal socrates.
```

```
Proof.
```

```
  apply (hm socrates).
  assumption.
```

```
Qed.
```

```
Print ms.
```

```
ms = hm socrates hs
    : mortal socrates
```

End Socrates.

\forall と \exists の間に De Morgan の法則がなりたつ。前回と同様に、 \exists を導出しようとしたときに classic を使わなければならない。

Section Laws.

```
Variables (A:Set) (P Q:A->Prop).
```

```
Lemma DeMorgan2 : (~ exists x, P x) -> forall x, ~ P x.
```

```
Proof.
```

```
  intros N x Px.
  apply N.
  exists x.
  apply Px.
```

```
Qed.
```

```

Theorem exists_or : (exists x, P x  $\vee$  Q x) -> (exists x, P x)  $\vee$  (exists x, Q x).
Proof.
  intros H.
  destruct H as [x [p|q]].
  left. exists x. assumption.
  right. exists x. assumption.
Qed.

```

(* 中まで破壊 *)

```

Hypothesis classic : forall P,  $\sim\sim$ P -> P.

```

```

Lemma DeMorgan2' : ( $\sim$  forall x, P x) -> exists x,  $\sim$  P x.
Proof.

```

```

  intros np.
  apply classic.
  intros nen.
  apply np; clear np.
  intros a; apply classic.
  intros np.
  apply nen.
  exists a; assumption.
Qed.

```

End Negation.

練習問題 1.1 以下の定理を *Coq* で証明せよ.

```

Section Coq3.

```

```

  Variable A : Set.
  Variable R : A -> A -> Prop.
  Variables P Q : A -> Prop.

```

```

  Theorem exists_postpone :
    (exists x, forall y, R x y) -> (forall y, exists x, R x y).
  Theorem or_exists : (exists x, P x)  $\vee$  (exists x, Q x) -> exists x, P x  $\vee$  Q x.

```

```

  Hypothesis classic : forall P,  $\sim\sim$ P -> P.
  Theorem remove_c : forall a,
    (forall x y, Q x -> Q y) ->
    (forall c, ((exists x, P x) -> P c) -> Q c) -> Q a.

```

```

End Coq3.

```

2 帰納法

Coq でデータ型を定義すると、自動的に帰納法の原理が生成される。

```

Module MyNat.
Inductive nat : Set := 0 : nat | S : nat -> nat.
nat is defined
nat_rect is defined
nat_ind is defined
nat_rec is defined

```

```

Check nat_ind.
nat_ind
  : forall P : nat -> Prop,
    P 0 ->
    (forall n : nat, P n -> P (S n)) ->

```

```
forall n : nat, P n
```

もっと分かりやすく書くと、`nat_ind`の型は $\forall P, P 0 \rightarrow (\forall n, P n \rightarrow P (S n)) \rightarrow (\forall n, P n)$ である。即ち P は0でなりたち、任意の n について P が n でなりたてば、 $n+1$ でもなりたつことが証明できれば、任意の n について P がなりたつ。

ちなみに、`nat_rec`の定義を見ると、

```
Check nat_rec.
nat_rec
  : forall P : nat -> Set,
    P 0 ->
    (forall n : nat, P n -> P (S n)) ->
    forall n : nat, P n
```

P がPropではなくSetを返すこと以外、全く同じである。

本当の定義を見ると、

```
Print nat_rect.
nat_rect =
fun (P : nat -> Type) (f : P 0) (f0 : forall n : nat, P n -> P (S n)) =>
fix F (n : nat) : P n :=
  match n as n0 return (P n0) with
  | 0 => f
  | S n0 => f0 n0 (F n0)
end
```

実は普通の再帰関数同様、`fix`と`match`を使って定義されている。

```
End MyNat.
Definition plus' : nat -> nat -> nat.
  intros m n.
  induction m.
  exact n.
  exact (S IHm).
Defined.
Print plus'.
fun m n : nat => nat_rec (fun _ : nat => nat) n (fun _ IHm : nat => S IHm) m
Eval compute in plus' 2 3.
= 5
```

(* 普通の nat に戻る *)

(* n を返す *)

(* 帰納法によって得られた IHm の後者を返す *)

(* 計算を可能にするために Defined で閉じる *)

(* 計算する *)

(* 反射率を調べる *)

(* 代入を行う *)

(* apply eq_refl と同じ *)

練習問題 2.1 以下の定理を証明せよ。

```
Theorem plus_0 : forall n, n + 0 = n.
Theorem plus_m_Sn : forall m n, m + (S n) = S (m + n).
Theorem plus_comm : forall m n, m + n = n + m.
Theorem plus_distr : forall m n p, (m + n) * p = m * p + n * p.
Theorem mult_assoc : forall m n p, m * (n * p) = (m * n) * p.
```