

# Coq の論理

## 1 プログラムの型付け

**型**  $\tau, \theta ::= \text{nat} \mid Z \mid \dots \mid \theta \rightarrow \tau \mid \tau \times \theta$  データ型, 関数型, 直積

**型判定**  $\Gamma \vdash M : \tau$   $\Gamma = x_1 : \tau_1, \dots, x_n : \tau_n$  という仮定のもとで,  $M$  が型  $\tau$  をもつ.

**型付け規則** Coq の式は以下の型付け規則によって型付けされる.

<b>変数</b>	$\Gamma \vdash x : \tau$ ( $x : \tau$ は $\Gamma$ に含まれる)	<b>定義</b>	$\frac{\Gamma \vdash M : \theta \quad \Gamma, x : \theta \vdash N : \tau}{\Gamma \vdash \text{let } x := M \text{ in } N : \tau}$
<b>抽象</b>	$\frac{\Gamma, x : \theta \vdash M : \tau}{\Gamma \vdash \text{fun } x : \theta \Rightarrow M : \theta \rightarrow \tau}$	<b>不動点</b>	$\frac{\Gamma, f : \theta \rightarrow \tau, x : \theta \vdash M : \tau}{\Gamma \vdash \text{fix } f (x : \theta) := M : \theta \rightarrow \tau}$
<b>適用</b>	$\frac{\Gamma \vdash M : \theta \rightarrow \tau \quad \Gamma \vdash N : \theta}{\Gamma \vdash M N : \tau}$	<b>直積</b>	$\frac{\Gamma \vdash M : \tau \quad \Gamma \vdash N : \theta}{\Gamma \vdash (M, N) : \tau \times \theta}$
	<b>射影</b>		$\Gamma \vdash \text{fst} : \tau \times \theta \rightarrow \tau \quad \Gamma \vdash \text{snd} : \tau \times \theta \rightarrow \theta$

**型付けの例**

$$\frac{\frac{\frac{\Gamma, x : \text{nat} \vdash S : \text{nat} \rightarrow \text{nat} \quad \Gamma, x : \text{nat} \vdash x : \text{nat}}{\Gamma, x : \text{nat} \vdash S x : \text{nat}} \text{適用}}{\Gamma \vdash \text{fun } x : \text{nat} \Rightarrow S x : \text{nat} \rightarrow \text{nat}} \text{抽象}}{\Gamma \vdash (\text{fun } x : \text{nat} \Rightarrow S x) O : \text{nat}} \text{適用} \quad \Gamma \vdash O : \text{nat}}$$

## 2 命題論理

**論理式** 論理式は以下の結合子から定義される.

$P, Q ::=$	$\text{True} \mid \text{False}$	定数
	$A$	論理変数
	$P \supset Q$	含意
	$P \wedge Q$	論理積
	$P \vee Q$	論理和

否定はないが, 便宜のために  $\neg P = P \supset \text{False}$  とおく.

**導出規則** 自然演繹体系では真の論理式は以下の規則より導出される.

$\Delta$  を論理式の集合とする.  $\text{True}$  は常に  $\Delta$  に含まれる.

<b>公理</b>	$\Delta \vdash P$ ( $P$ は $\Delta$ に含まれる)	<b><math>\wedge</math> 導入</b>	$\frac{\Delta \vdash P \quad \Delta \vdash Q}{\Delta \vdash P \wedge Q}$
<b><math>\supset</math> 導入</b>	$\frac{\Delta, P \vdash Q}{\Delta \vdash P \supset Q}$	<b><math>\wedge</math> 除去</b>	$\frac{\Delta \vdash P \wedge Q \quad \Delta \vdash P \wedge Q}{\Delta \vdash P \quad \Delta \vdash Q}$
<b><math>\supset</math> 除去</b>	$\frac{\Delta \vdash P \quad \Delta \vdash P \supset Q}{\Delta \vdash Q}$	<b><math>\vee</math> 導入</b>	$\frac{\Delta \vdash P \quad \Delta \vdash Q}{\Delta \vdash P \vee Q \quad \Delta \vdash P \vee Q}$
<b>背理法</b>	$\frac{\Delta, \neg P \vdash \text{False}}{\Delta \vdash P}$	<b><math>\vee</math> 除去</b>	$\frac{\Delta \vdash P \vee Q \quad \Delta, P \vdash R \quad \Delta, Q \vdash R}{\Delta \vdash R}$

**恒真式** 命題論理の恒真式は *True* だけを仮定して導出できる式である。

例えば、 $P \supset P \wedge P$  や  $P \supset (P \supset Q) \supset Q$  は恒真式である。それぞれの導出を以下に示す。

$$\frac{\frac{P \vdash P \quad P \vdash P \quad (\text{公理})}{P \vdash P \wedge P} (\wedge \text{導入})}{\vdash P \supset P \wedge P} (\supset \text{導入}) \quad \frac{\frac{P, P \supset Q \vdash P \quad P, P \supset Q \vdash P \supset Q \quad (\text{公理})}{P, P \supset Q \vdash Q} (\supset \text{導入})}{\vdash P \supset (P \supset Q) \supset Q} (\supset \text{導入})$$

### 3 命題と型の対応

**カリリー・ハワード同型**により、命題論理と型理論 (型付λ計算) が対応している。

具体的には、以下のような対応が見られる。

命題 (論理式)	型
証明 (導出)	プログラム
仮定 $\Delta$	型環境 $\Gamma$
$\supset$	$\rightarrow$
$\wedge$	*

導出規則と型付け規則も基本的には1対1で対応している。それぞれの体系を少し修正すると以下の定理がなりたつ。

**定理 1 (Curry-Howard 同型)** ある同型  $\langle \_ \rangle : \text{命題} \rightarrow \text{型}$  が存在し、任意の  $\Delta$  と  $P$  について、導出  $\Pi$  より  $\Delta \vdash P$  が示せるならば、 $\Pi$  からプログラム  $M$  が作れ、 $\langle \Delta \rangle \vdash M : \langle P \rangle$ 。また、任意の  $\Gamma, M, \tau$  について型理論で  $\Gamma \vdash M : \tau$  が導出できれば、命題論理において  $\langle \Gamma \rangle^{-1} \vdash \langle \tau \rangle^{-1}$  が導出できる。

修正の内容は二種類ある。

まず、上の**不動点**の規則は矛盾を生んでしまう。具体的には、 $\theta = \text{True}$  と  $\tau = \text{False}$  にすると、以下の導出が可能になる。

$$\frac{\Gamma, f : \text{True} \rightarrow \text{False}, x : \text{True} \vdash f x : \text{False}}{\Gamma \vdash \text{fix } f (x:\theta) := f x : \text{True} \rightarrow \text{False}}$$

しかし、Coq の本当の**不動点**の規則はさらに  $f$  が  $x$  より小さな引数に適用されることを求めているので、この矛盾が実際には起きない。本当の規則が複雑なのでここには書かない。

もう一つは、**背理法**に対する規則は Coq の型体系にはない。それは Coq は通常の命題論理 (**古典論理**) に基いているのではなく、それと少し異なる**直感主義論理**に基いているからである。もしも命題論理を直感主義にするならば、背理法を以下の**矛盾**という規則に置き換えればいい。

$$\text{矛盾} \quad \frac{\Delta \vdash \text{False}}{\Delta \vdash P}$$

要するに、矛盾 (*False*) が証明できれば、何でも証明できるようにする。古典論理では背理法よりそれが導出できるが、背理法のない直観主義論理ではこの新しい規則が必要になる。

Coq の論理はこの直観主義論理とちょうど一致する。メリットとして、全ての証明が計算的な意味を持つ—証明は関数である。

しかし、逆に Coq の中で古典論理の証明をしたいときもある。ほとんどの定理は背理法なしで証明できるものの、証明できない定理もある上、単に背理法が便利なきときもある。そのとき、Coq の論理に新しい公理として以下の規則を導入すればいい。

$$\neg\neg\text{除去} \quad \Delta \vdash \neg\neg P \supset P$$

この公理と抽象を組合せると背理法が導出可能になる。

## 4 Coqで定理の証明

前述の Curry-Howard 同型のおかげで、Coq の中で直接に命題を書くことができる。その型を満すプログラムが見付かれれば、定理になる。

**変数宣言** まずは、準備として論理変数の宣言を行う。Section というコマンドを使うと、局所的な論理変数が宣言できるようになる。宣言自体は Variables コマンドを使う。そして、宣言範囲が終ると End コマンドでセクションを閉じる。

```
Section Koushin.  
  
Variables P Q : Prop.  
P is assumed  
Q is assumed
```

論理式自身は型であると先に説明したが、通常の型の型だった Set と異なり、論理式の型は Prop になる。普段はあまり影響はないが、区別すると便利なことができる。

### 命題と証明プログラム

まず、前の二つの恒真式を証明してみよう。  
2つ目は関数適用だけなので、簡単にできる。

```
Theorem modus_ponens : P -> (P -> Q) -> Q.      (* 名前を付けなければならない *)  
Proof (fun p pq => pq p).  
modus_ponens is defined
```

```
Print modus_ponens.                               (* 実際には関数定義と変わらない *)  
modus_ponens = fun (p : P) (pq : P -> Q) => pq p  
              : P -> (P -> Q) -> Q
```

しかし、一つ目ではデータの直積ではなく、命題の論理積を使ったので、作り方を調べなければならぬ。

```
SearchPattern (_ /\ _).                          (* 論理積を返す関数(定理)を調べる *)  
andb_prop: forall a b : bool, (a && b)%bool = true -> a = true /\ b = true  
conj: forall A B : Prop, A -> B -> A /\ B  
iff_and: forall A B : Prop, (A <-> B) -> (A -> B) /\ (B -> A)
```

この中では、conj が期待の操作をしている。

```
Theorem and_self : P -> P /\ P.  
Proof (fun x => conj x x).  
and_self is defined
```

### 作戦 (tactic) の利用

上のように、プログラムを与えることで定理を証明することができる。しかし、複雑な定理になると、途中で出て来る命題が煩雑になり、正しいプログラムを書くのが至難の技になる。

通常は、定理は関数と違う定義方法を使う。証明モードに入り、作戦 (tactic) によって証明を構築していく。各 tactic は導出規則と対応している。

```
Theorem modus_ponens' : P -> (P -> Q) -> Q.      (* 異なる名前にする *)  
1 subgoal                                         (* 証明の状況が表示される *)  
  
P : Prop  
Q : Prop  
=====
```

$P \rightarrow (P \rightarrow Q) \rightarrow Q$

Proof.

intros p pq.

(\* 仮定に名前を付ける (抽象) \*)

$p : P$

$pq : P \rightarrow Q$

=====

$Q$

apply pq.

(\* 目標を関数 pq の結果とみなす (適用) \*)

$p : P$

$pq : P \rightarrow Q$

=====

$P$

assumption.

*Proof completed.*

Qed.

*modus\_ponens' is defined*

実際の証明をもう一度みよう.

Theorem modus\_ponens' :  $P \rightarrow (P \rightarrow Q) \rightarrow Q$ .

Proof.

intros p pq.

apply pq.

assumption.

Qed.

and\_self について同じことをする.

Theorem and\_self' :  $P \rightarrow P \wedge P$ .

Proof.

intros p.

1 subgoal

$p : P$

=====

$P \wedge P$

split.

2 subgoals

(\* 論理積の導入 ( $\wedge$  導入) \*)

(\* 前提が二つある \*)

$p : P$

=====

$P$

subgoal 2 is:

$P$

assumption.

(\* 順番に解いていく \*)

1 subgoal

$p : P$

=====

$P$

assumption.

Qed.

*and\_self' is defined*

```
Print and_self'.
and_self' = fun p : P => conj p p
           : P -> P /\ P
```

(\* 実際の定義は前と変わらない \*)

## セクションを閉じる

End Koushin.

```
Print and_self.
and_self =
fun (P : Prop) (x : P) => conj x x
   : forall P : Prop, P -> P /\ P
```

(\* 必要な変数が定義に挿入される \*)

## 否定に関する定理

証明状態の表示が作戦を読みにくくするので、これ以降は省くことにする。自分で Coq の中で実行して、確認して下さい。

Section Negation.

Variables P Q : Prop.

Theorem DeMorgan :  $\sim (P \vee Q) \rightarrow \sim P \wedge \sim Q$ .

Proof.

```
intros npq.
split; intros q.
  apply npq.
  left.
  assumption.
elim npq.
right.
assumption.
```

(\* ; で両方の subgoal について intros q を行う \*)

(\*  $\vee$  導入の左を使う \*)

Qed.

*DeMorgan is defined*

しかし、双対的な定理  $(\neg(P \wedge Q) \supset \neg P \vee \neg Q)$  は直観主義論理ではなりたたない。Hypothesis コマンドによって二重否定の除去を仮定すると証明できる。ちなみに、Hypothesis コマンドは Variables の異名でしかなくて、動作は全く同じである。

```
Hypothesis classic : forall P,  $\sim\sim P \rightarrow P$ .
classic is assumed
```

(\* 任意の P について \*)

Theorem DeMorgan' :  $\sim (P \wedge Q) \rightarrow \sim P \vee \sim Q$ .

Proof.

```
intros npq.
apply classic.
intro nnpq.
apply npq.
clear npq.
split; apply classic.
  intros np.
  apply nnpq.
  left.
  assumption.
intros np; apply nnpq; right; assumption.
```

(\* 不要な仮定を忘れる \*)

Qed.

*DeMorgan' is defined*

End Negation.

### 仮定を破壊する

Coq の帰納的データ型に対して、値を破壊しながら中身を取り出すという tactic が便利である。直接に対応する論理規則はないが、当然ながら他の論理規則から同じ結果を導くことは可能である。

Section Destruct.

Variables P Q : Prop.

Theorem and\_comm : P ∧ Q -> Q ∧ P.

Proof.

intros pq.

destruct pq as [p q].

(\* 中身を取り出す \*)

split; assumption.

(\* 一気に終わらせる \*)

Qed.

*and\_comm is defined*

Theorem or\_comm : P ∨ Q -> Q ∨ P.

Proof.

intros pq.

destruct pq as [p | q].

(\* 場合が二つある \*)

right; assumption.

left; assumption.

Qed.

*or\_comm is defined*

End Destruct.

### 論理規則と tactic の対応

論理規則	型付け規則	作戦
公理	変数	assumption
⊃ 導入	抽象	intros <i>h</i>
⊃ 除去	適用	apply <i>h</i>
矛盾		elimtype False
∧ 導入	直積	split
∧ 除去	射影	destruct <i>h</i> as [ <i>h</i> <sub>1</sub> <i>h</i> <sub>2</sub> ]
∨ 導入	直和	left, right
∨ 除去	match	destruct <i>h</i> as [ <i>h</i> <sub>1</sub>   <i>h</i> <sub>2</sub> ]

練習問題 4.1 以下の定理を Coq で証明せよ。

Section Coq2.

Variables P Q R : Prop.

Theorem imp\_trans : (P -> Q) -> (Q -> R) -> P -> R.

Theorem not\_false : ~False.

Theorem double\_neg : P -> ~~P.

Theorem contraposition : (P -> Q) -> ~Q -> ~P.

Theorem and\_assoc : P ∧ (Q ∧ R) -> (P ∧ Q) ∧ R.

Theorem and\_distr : P ∧ (Q ∨ R) -> (P ∧ Q) ∨ (P ∧ R).

Theorem absurd : P -> ~P -> Q.

End Coq2.