Jacques Garrigue, 2015 年 10 月 14 日

Coqの論理

1 プログラムの型付け

型 $\tau, \theta ::= \mathsf{nat} \mid \mathsf{Z} \mid \ldots \mid \theta \to \tau \mid \tau \times \theta$ データ型, 関数型, 直積

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

変数 $\Gamma \vdash x : \tau \quad (x : \tau \text{は} \Gamma \text{に含まれる})$

定義 $$\frac{\Gamma \vdash M : \theta \quad \Gamma, x : \theta \vdash N : \tau}{\Gamma \vdash \mathtt{let}\ x\ \mathtt{:=}\ M\ \mathtt{in}\ N : \tau}$$

抽象 $$\frac{\Gamma, x : \theta \vdash M : \tau}{\Gamma \vdash \mathtt{fun}\ x{:}\theta \Rightarrow M : \theta \to \tau}$$

不動点 $$\frac{\Gamma, f : \theta \to \tau, x : \theta \vdash M : \tau}{\Gamma \vdash \mathtt{fix}\ f\ (x{:}\theta)\ \mathtt{:=}\ M : \theta \to \tau}$$

適用 $$\frac{\Gamma \vdash M : \theta \to \tau \quad \Gamma \vdash N : \theta}{\Gamma \vdash M\ N : \tau}$$

直積 $$\frac{\Gamma \vdash M : \tau \quad \Gamma \vdash N : \theta}{\Gamma \vdash (M, N) : \tau \times \theta}$$

射影 $\Gamma \vdash \mathsf{fst} : \tau \times \theta \to \tau \qquad \Gamma \vdash \mathsf{snd} : \tau \times \theta \to \theta$

型付けの例

$$\cfrac{\cfrac{\cfrac{\Gamma, x : \mathsf{nat} \vdash \mathsf{S} : \mathsf{nat} \to \mathsf{nat} \quad \Gamma, x : \mathsf{nat} \vdash x : \mathsf{nat}}{\Gamma, x : \mathsf{nat} \vdash \mathsf{S}\ x : \mathsf{nat}}\ \text{適用}}{\Gamma \vdash \mathtt{fun}\ x{:}\mathsf{nat} \Rightarrow \mathsf{S}\ x : \mathsf{nat} \to \mathsf{nat}}\ \text{抽象} \qquad \Gamma \vdash \mathsf{O} : \mathsf{nat}}{\Gamma \vdash (\mathtt{fun}\ x{:}\mathsf{nat} \Rightarrow \mathsf{S}\ x)\ \mathsf{O} : \mathsf{nat}}\ \text{適用}$$

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．

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

否定はないが，便宜のために $\neg P = P \supset \mathit{False}$ 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．
Δ を論理式の集合とする．True は常に Δ に含まれる．

公理 $\Delta \vdash P \quad (P \text{は} \Delta \text{に含まれる})$

$\wedge$ 導入 $$\frac{\Delta \vdash P \quad \Delta \vdash Q}{\Delta \vdash P \wedge Q}$$

$\supset$ 導入 $$\frac{\Delta, P \vdash Q}{\Delta \vdash P \supset Q}$$

$\wedge$ 除去 $$\frac{\Delta \vdash P \wedge Q}{\Delta \vdash P} \qquad \frac{\Delta \vdash P \wedge Q}{\Delta \vdash Q}$$

$\supset$ 除去 $$\frac{\Delta \vdash P \quad \Delta \vdash P \supset Q}{\Delta \vdash Q}$$

$\vee$ 導入 $$\frac{\Delta \vdash P}{\Delta \vdash P \vee Q} \qquad \frac{\Delta \vdash Q}{\Delta \vdash P \vee Q}$$

背理法 $$\frac{\Delta, \neg P \vdash \mathit{False}}{\Delta \vdash P}$$

$\vee$ 除去 $$\frac{\Delta \vdash P \vee Q \quad \Delta, P \vdash R \quad \Delta, Q \vdash R}{\Delta \vdash R}$$

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．

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

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 Δ	型環境 Γ
$\supset$	$\rightarrow$
$\wedge$	$*$

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 $\langle \_ \rangle$: 命題 $\rightarrow$ 型 が存在し，任意の Δ と P について，導出 Π より $\Delta \vdash P$ が示せるならば，Π からプログラム M が作れ，$\langle \Delta \rangle \vdash M : \langle P \rangle$．また，任意の Γ, M, τ 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，$\theta = \mathit{True}$ と $\tau = \mathit{False}$ にすると，以下の導出が可能になる．

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

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．

$$\textbf{矛盾} \quad \frac{\Delta \vdash \mathit{False}}{\Delta \vdash P}$$

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ない．

```
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 -> (P -> Q) -> Q

Proof.
  intros p pq.                              (* 仮定に名前を付ける（抽象）*)

  p : P
  pq : P -> Q
  ============================
   Q

  apply pq.                     (* 目標を関数 pq の結果とみなす（適用）*)

  p : P
  pq : P -> 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 -> (P -> Q) -> 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 -> P /\ P.
Proof.
  intros p.
1 subgoal

  p : P
  ============================
   P /\ P

  split.                                    (* 論理積の導入（∧ 導入）*)
2 subgoals                                          (* 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'.                              (* 実際の定義は前と変わらない *)
and_self' = fun p : P => conj p p
     : P -> P /\ P
```

セクションを閉じる

```
End Koushin.

Print and_self.
and_self =                                    (* 必要な変数が定義に挿入される *)
fun (P : Prop) (x : P) => conj x x
     : forall P : Prop, P -> P /\ P
```

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 : ~ (P \/ Q) -> ~ P /\ ~ Q.
Proof.
  unfold not.                                 (* ~ の定義を展開する *)
  intros npq.
  split; intros q.                (* ; で両方の subgoal について intros q を行う *)
    apply npq.
    left.                                     (* ∨ 導入の左を使う *)
    assumption.
  apply npq.
  right.
  assumption.
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, ~~P -> P.      (* 任意の P について *)
classic is assumed

Theorem DeMorgan' : ~ (P /\ Q) -> ~ P \/ ~ Q.
Proof.
  intros npq.
  apply classic.
  intro nnpq.
  apply npq.
  clear npq.                                  (* 不要な仮定を忘れる *)
  split; apply classic.
    intros np.
    apply nnpq.
    left.
    assumption.
  intros np; apply nnpq; right; assumption.
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` h
⊃ 除去	適用	`apply` h
矛盾		`elimtype False`
∧ 導入	直積	`split`
∧ 除去	射影	`destruct` h `as [`h_1 h_2`]`
∨ 導入	直和	`left, right`
∨ 除去	`match`	`destruct` h `as [`h_1 `|` h_2`]`

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.
```