Jacques Garrigue, 2020 年 10 月 7 日

Coq/SSReflect の論理

1 Coq の論理

Coq は Calculus of Inductive Constructions (CIC) という型理論に基いている．型理論は型付き λ 計算という計算体系と直観主義論理の証明論の間の同型 (カリー・ハワード対応) に基いた論理体系である．CIC は特に表現力が高い (一階述語論理や高階論理を含んでいる) が，まずその命題論理の部分から見ていくことにする

1.1 直観主義の命題論理

論理式　論理式は以下の結合子から定義される．

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

否定はないが，便宜のために $\neg P = P \to \mathit{False}$ 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　$\Delta \vdash P$　(P は Δに含まれる)

$\to$ **導入**　$\dfrac{\Delta, P \vdash Q}{\Delta \vdash P \to Q}$

$\to$ **除去**　$\dfrac{\Delta \vdash P \quad \Delta \vdash P \to Q}{\Delta \vdash Q}$

矛盾　$\dfrac{\Delta \vdash \mathit{False}}{\Delta \vdash P}$

$\wedge$ **導入**　$\dfrac{\Delta \vdash P \quad \Delta \vdash Q}{\Delta \vdash P \wedge Q}$

$\wedge$ **除去**　$\dfrac{\Delta \vdash P \wedge Q}{\Delta \vdash P} \quad \dfrac{\Delta \vdash P \wedge Q}{\Delta \vdash Q}$

$\vee$ **導入**　$\dfrac{\Delta \vdash P}{\Delta \vdash P \vee Q} \quad \dfrac{\Delta \vdash Q}{\Delta \vdash P \vee Q}$

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

この導出規則から背理法や排中律が導けない．もしも古典論理の体系が欲しければ，**矛盾**の代わりに以下の規則のどちらかを使う．

背理法　$\dfrac{\Delta, \neg P \vdash \mathit{False}}{\Delta \vdash P}$　　$\neg\neg$**除去**　$\Delta \vdash \neg\neg P \to P$

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，$P \to P \wedge P$ や $P \to (P \to Q) \to Q$ は恒真式である．それぞれの導出を以下に示す．

$$\dfrac{\dfrac{\dfrac{\dfrac{P \wedge Q \vdash P \wedge Q}{P \wedge Q \vdash Q} \quad \dfrac{P \wedge Q \vdash P \wedge Q}{P \wedge Q \vdash P}}{P \wedge Q \vdash Q \wedge P}}{\vdash P \wedge Q \to Q \wedge P}}{\begin{array}{r}(\text{公理}) \\ (\wedge\text{除去}) \\ (\wedge\text{導入}) \\ (\to \text{導入})\end{array}} \qquad \dfrac{\dfrac{\dfrac{P, P \to Q \vdash P \quad P, P \to Q \vdash P \to Q}{P, P \to Q \vdash Q}}{P \vdash (P \to Q) \to Q}}{\vdash P \to (P \to Q) \to Q}\begin{array}{r}(\text{公理}) \\ (\to \text{除去}) \\ (\to \text{導入}) \\ (\to \text{導入})\end{array}$$

1.2 Coq での証明

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．Coq の出力をイタリック体で表示している．

```
Section Koushin.

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

変数を宣言するときにその型も書かなければならないが，論理式の型は Prop になる．

命題と証明

まず，二つ目の恒真式を証明してみよう．定理を宣言すると証明モードに移る．

```
Theorem modus_ponens : P -> (P -> Q) -> Q.      (* 名前を付けなければならない *)
1 subgoal                                          (* 証明の状況が表示される *)
  P, Q : Prop
  ============================
   P -> (P -> Q) -> Q
```

線の上は Δ の中身，下は結論だが，今は Δ に変数の宣言しかない．P と Q が命題であるという宣言であり，P や Q が成り立つという仮定ではない．$\rightarrow$ **導入**にあたる Coq のタクティック (証明命令) は intros である．$\rightarrow$ **除去**は apply．**公理**は assumption.

```
Proof.
  intros p pq.                                  (* 仮定に名前を付ける（導入）*)
1 subgoal
  p : P
  pq : P -> Q
  ============================
   Q

  apply pq.                                  (* 目標を仮定 pq の帰結とみなす（除去）*)
1 subgoal
  p : P
  pq : P -> Q
  ============================
   P

  assumption.                                                  (*（公理）*)
  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.
```

証明が長い訳ではないが，少し違和感がある．直感的に二つ目の前提を一つ目に適用すれば結論が出るはずなのに，逆にそれを結論に「適用」している．証明を結論から作っていくので，考

え方が逆になってしまい，証明が長くなると追いにくい．

2 SSReflect での証明

上の自然演繹体系は Coq の中の論理によく合っているが，証明の構築には不便である．証明の構築は，証明したい命題と仮定を下に書き，証明木を延して行く形で行うが，自然演繹の場合では，規則は結論 (右側) に対するものばかりで，その結論を変形させる「逆算」で行うことになる．通常の数学の証明は仮定から性質を導く「前進」方向で行うことが多い．Gentzen のシークエント計算が結論を扱う「右規則」とは別に仮定を扱う「左規則」も持っている．ここでは，そういう左規則を自然演繹に追加する形で SSReflect の証明方法を反映した論理体系を作る．

2.1 SSReflect の論理

新しい判定の形は

$$\Delta; P \vdash Q$$

Δ には名前付きの仮定が入っている．P は焦点と言い，左 (L) 規則の対象となる．焦点は空白でもいい．Q は結論といい，右 (R) 規則の対象である．

Focus $\dfrac{\Delta; P \vdash Q}{\Delta; \vdash P \to Q}$

Unfocus $\dfrac{\Delta; \vdash P \to Q}{\Delta; P \vdash Q}$

Intro $\dfrac{\Delta, h : P; \vdash Q}{\Delta; P \vdash Q}$

Revert $\dfrac{\Delta; P \vdash Q}{\Delta, h : P; \vdash Q}$

Apply-R $\dfrac{\Delta; \vdash P_1 \quad \ldots \quad \Delta; \vdash P_n}{\Delta; P_1 \to \ldots \to P_n \to Q \vdash Q}$

Apply-L1 $\dfrac{\Delta; \vdash P \to Q \quad \Delta; Q \vdash R}{\Delta; P \vdash R}$

Apply-L2 $\dfrac{\Delta; \vdash P \quad \Delta; Q \vdash R}{\Delta; P \to Q \vdash R}$

公理 $\Delta; P \vdash P$

矛盾 $\Delta; \mathit{False} \vdash P$

カット $\dfrac{\Delta; \vdash P \quad \Delta; P \vdash Q}{\Delta; \vdash Q}$

∧-R $\dfrac{\Delta; \vdash P \quad \Delta; \vdash Q}{\Delta; \vdash P \wedge Q}$

∧-L $\dfrac{\Delta; P \vdash Q \to R}{\Delta; P \wedge Q \vdash R}$

∨-R $\dfrac{\Delta; \vdash P}{\Delta; \vdash P \vee Q} \quad \dfrac{\Delta; \vdash Q}{\Delta; \vdash P \vee Q}$

∨-L $\dfrac{\Delta; P \vdash R \quad \Delta; Q \vdash R}{\Delta; P \vee Q \vdash R}$

規則は下から上へと読む．**Focus** は結論の前提を焦点に移す．**Unfocus** は逆に焦点された命題を結論の前提に戻す．SSReflect ではこの二つの規則は暗に使われ，焦点と結論の前提が区別されない．**Intro** は焦点に名前を付けて仮定に移す．**Revert** は仮定を焦点に戻す．基本的にはこの 4 つの規則が仮定を動かしているいるだけだ．**Apply-R** は焦点を使って結論を証明するが，その際焦点の全ての前提を証明する必要が新たに生じる．**Apply-L1** は補題 $P \to Q$ を使って焦点を変える．**Apply-L2** は逆に補題 P を使って焦点の前提をみたす．この 3 つの規則を右側の**カット**で代用することができるので，論理的には必要ではないが，証明を自然に進めるのに使う．右側の規則は右規則である導入が自然演繹と変らないが，除去に当たる左規則やカットが焦点を利用している．特別な場合として，**∧-L** では焦点に P しか入らないので，Q が前提に戻される．

2.2 SSReflect のタクティック

SSReflect の論理は多くの規則をもっているが，Coq と違い各規則が独立したタクティックに対応している訳ではない．ほとんどのものが「move+修飾子」という形で書ける．

もう一回 modus_ponens を証明する．

```
Reset modus_ponens.                          (* module_ponens 以降の定義を忘れる *)
Require Import ssreflect.                   (* ssreflect のタクティック言語を使う *)

Theorem modus_ponens : P -> (P -> Q) -> Q.
Proof.
  move=> p.                                                    (* Intro *)
1 subgoal
  p : P
  ============================
  (P -> Q) -> Q

  move/(_ p).                                                  (* Apply-L2 *)
1 subgoal
  p : P
  ============================
  Q -> Q

  done.                                                        (* 公理や矛盾 *)
No more subgoals.                                              (* 証明完了 *)
Restart.                                                       (* 証明をやりなおす *)
  by move => p /(_ p).                                         (* 全てを一行にまとめる *)
Qed.
```

move の => と / は一緒にまとめられ，さらに by を他のタクティックの前に書くと，後に done を加えたときと同じ意味になる．

もう一つの恒真も試してみよう．

```
Theorem and_comm : P /\ Q -> Q /\ P.
Proof.
  move=> [] p q.                                    (* ∧-L, Intro, Intro *)
  p : P
  q : Q
  ============================
  Q /\ P

  split.                                                       (* ∧-R *)
2 subgoals

  P, Q : Prop
  p : P
  q : Q
  ============================
  Q

subgoal 2 is:
 P

  done. done.
No more subgoals.

Restart.
  by move=> [p q]; split.        (* 名前を中に書く，タクティックを「;」でつなぐ *)
```

```
Qed.
```

恒真色々　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．

```
Theorem or_comm : P \/ Q -> Q \/ P.
Proof.
  move=> [p|q].              (* (∨-L; 証明木が分岐するとき，縦棒「|」で分ける *)
    by right.                                                  (* ∨-R2 *)
  by left.                                                     (* ∨-R1 *)
Qed.

Theorem DeMorgan : ~ (P \/ Q) -> ~ P /\ ~ Q.
Proof.
  rewrite /not.                                          (* 定義を展開する *)
    P, Q : Prop
    ============================
    (P ∨ Q -> False) -> (P -> False) ∧ (Q -> False)
  move=> npq.
  split=> [p|q].                          (* 全てのタクティックで「=>」が使える *)
    apply: npq.                                          (* (Apply-R) *)
    by left.
  apply: npq.
  by right.
Qed.
End Koushin.                                          (* Section を閉じる *)
```

論理和に関する De Morgan の法則が証明できた．しかし，双対的な法則 $(\neg(P \wedge Q) \supset \neg P \vee \neg Q)$ は直観主義論理ではなりたたない．二重否定の除去か排中律を仮定する必要がある．それらの同値性を証明した上でその定理を証明する．

```
Section Classic.
Definition Classic := forall P : Prop, ~ ~ P -> P.          (* ¬¬ 除去の定義 *)
Definition EM := forall P : Prop, P \/ ~ P.                  (* 排中律の定義 *)

Lemma Classic_is_EM : Classic <-> EM.                 (* ¬¬ 除去と排中律が同値 *)
Proof.
rewrite /Classic /EM.                                          (* 定義の展開 *)
split => [classic | em] P.                    (* A <-> B := A -> B /\ B -> A *)
  - apply: (classic) => nEM.          (* classic を仮定から消さずに焦点におく *)
    have p : P.                                              (* (カット) *)
      apply: classic => np.
      apply: nEM. by right.
    apply: nEM. by left.
  - move: (em P) => [].                   (* P についての排中律で場合分けをする *)
    + done.
    + move => np /(_ np). done.
Qed.
```

Definition は項や命題を定義する．ここでは二重否定除去と排中律の言明を定義するのに使う．forall P : Prop はこの言明が任意の命題に対して使えることを表している．

証明の中の「-」(または「+」や「*」) は場合分けの構造を分かりやすくするために使う．一時的に他のゴールが見えなくなる．

```
Variables P Q : Prop.

Theorem DeMorgan' : Classic -> ~ (P /\ Q) -> ~ P \/ ~ Q.
```

```
Proof.
  move=> /Classic_is_EM em npq.                    (* Apply-L1, Intro, Intro *)
  move: (em P) => [p|np].                                (* 排中律で場合分け *)
  - move: (em Q) => [q|nq].
    + elim: npq.                                                    (* 矛盾 *)
      by split.
    + by right.
  - by left.
Qed.
End Classic.                                              (* Section を閉じる *)
Check DeMorgan'.                                    (* 命題の言明を確認する *)
DeMorgan'
     : forall P Q : Prop, Classic -> ~ (P /\ Q) -> ~ P \/ ~ Q
```

Section から出ると，使われた命題変数が各定理の言明に自動で追加される．

論理規則と tactic の対応

論理規則	作戦	論理規則	作戦
Intro	move => h	公理	done, by
Revert	move: h	矛盾	elim, suff: False
Apply-R	apply	$\wedge$-R	split
Apply-L1	move/e	$\wedge$-L	move => []
Apply-L2	move/(_ e)	$\vee$-R	left, right
カット	have: P, suff: P, move: (e)	$\vee$-L	move => [|]

h は仮定の名前，e は仮定を適用で組み合わせた証明式，P は論理式である．各タクティックの詳しい説明は次回 (述語論理) のときに述べる．

練習問題 2.1 以下の定理を Coq で証明せよ．

```
Section Coq1.
  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.
  Definition DM_rev := forall P Q, ~ (~P /\ ~Q) -> P \/ Q.
  Theorem DM_rev_is_EM : DM_rev <-> EM.
End Coq1.
```