「計算と論理」
Software Foundations
その10

五十嵐 淳
igarashi@kuis.kyoto-u.ac.jp

京都大学

January 8, 2013

本日のメニュー

Logic.v (Coq の論理) 続き

- 量化と含意
- 連言 (「かつ」)
- 選言 (「または」)
- **偽・矛盾**
  - **真**
- **否定 (「〜でない」)**
- **特称量化 (「ある x が存在して〜」)**
- **等号**
- **命題としての関係**
- **非形式的証明**

偽 (falsehood)

「偽」(⊥ とも書かれる) の (帰納的な) 定義

```
Inductive False : Prop := .
```

- コンストラクタがひとつもない定義
  - 偽の証明は存在しない
- 自然演繹における導出規則

$$\frac{}{\Gamma \vdash \bot : \mathtt{Prop}} \quad (\mathrm{P}\text{-}\bot)$$

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash P} \quad (\bot\text{-}\mathrm{E})$$

「偽」についての証明(1)

```
Check False_ind.

Theorem False_implies_nonsense :
  False -> 2 + 2 = 5.
Proof.
  intros contra.
  inversion contra.  Qed.
```

- inversion: **場合わけにより，コンストラクタの数に応じたサブゴール生成**
- **コンストラクタの数が0なので，サブゴールの数も0**
- **つまり証明完了!**

「偽」についての証明(2)

False を結論として導く唯一の手段は仮定の矛盾を指摘すること:

```
Theorem nonsense_implies_False :
  2 + 2 = 5 -> False.
Proof.
  intros contra. inversion contra.  Qed.
```

規則 ⊥-E (爆発則とも呼ぶ)

```
Theorem ex_falso_quodlibet : forall (P:Prop),
  False -> P.
Proof.
  intros P contra. inversion contra.  Qed.
```

本日のメニュー

Logic.v (Coq の論理)

- 量化と含意
- 連言 (「かつ」)
- 選言 (「または」)
- 偽・矛盾
- 否定 (「〜でない」)
  - 不等号 (等しくない)
- 特称量化 (「ある x が存在して〜」)
- 等号
- 命題としての関係
- 非形式的証明

否定

「P ではない」(¬P, ∼ P) の定義

```
Definition not (P:Prop) := P -> False.
 (* 「P ではない」 = P を仮定すると矛盾する *)

Notation "~ x" := (not x) : type_scope.
```

否定を使った証明(1)

否定の証明は(`False`を導くことになるので)少しコツが必要.

```
Theorem not_False : ~ False.
Proof.
  unfold not. intros H. inversion H.  Qed.

Theorem contradiction_implies_anything :
  forall P Q : Prop, (P /\ ~P) -> Q.
Proof.
  intros P Q H. inversion H as [HP HNA].
  unfold not in HNA.
  apply HNA in HP. inversion HP.  Qed.
```

否定を使った証明(2)

```
Theorem double_neg : forall P : Prop,
  P -> ~~P.
Proof.
  intros P H. unfold not.
  intros G. apply G. apply H.  Qed.
```

否定を使った証明(3)

文脈の移り変わりに注意!

```
Theorem five_not_even :
  ~ ev 5.
Proof.
  unfold not. intros Hev5.
  inversion Hev5 as [|n Hev3 Heqn].
  inversion Hev3 as [|n' Hev1 Heqn'].
  inversion Hev1.
Qed.
```

Coq の論理は古典論理ではない!

- **古典論理**: **真理値表で意味を与える論理**
- Coq **の論理は直観主義論理** (intuitionistic logic) **と呼ばれる**
- **古典論理では正しい命題でも成立しないものがある**
- **例**: **二重否定の除去**

```
Theorem classic_double_neg : forall P : Prop,
  ~~P -> P.
Proof.
  intros P H. unfold not in H.
  (* But now what? There is no way to
     "invent" evidence for [P]. *)
  Admitted.
```

直観主義論理では成立しない古典論理公理

- **パース則**: ((P -> Q) -> P) -> P
- **排中律**: P ∨ ¬P
  - **ただし，** ¬¬(P ∨ ¬P) **は成立**
- **ド・モルガン則 (の一部)**: ¬(P ∧ Q) -> ¬P ∨ ¬Q
- (P -> Q) -> (¬P ∨ Q)

不等号

x <> y は ¬(x = y) のこと:

```
Notation "x <> y" := (~ (x = y)) : type_scope.

Theorem not_false_then_true : forall b : bool,
  b <> false -> b = true.
Proof.
  intros b H. destruct b.
  Case "b = true". reflexivity.
  Case "b = false".
    unfold not in H.
    apply ex_falso_quodlibet. (* 定石 *)
    apply H. reflexivity.   Qed.
```

本日のメニュー

Logic.v (Coq の論理)

- 量化と含意
- 連言 (「かつ」)
- 選言 (「または」)
- 偽・矛盾
- 否定 (「〜でない」)
- 特称量化 (「ある x が存在して〜」)
- 等号
- 命題としての関係
- 非形式的証明

特称量化

「型 X の要素 x が存在して P」($\exists x : X.P$) の帰納的定義:

```
Inductive ex (X:Type) (P : X->Prop) : Prop :=
  ex_intro : forall (witness:X),
               P witness -> ex X P.

Notation "'exists' x : X , p" :=
  (ex _ (fun x:X => p))
  (at level 200, x ident, right associativity)
  : type_scope.
```

- **直観**: `ex X P` **の証拠は型 X の要素 witness と P witness の証拠の組**

自然演繹における導出規則

$$\frac{\Gamma, x : T \vdash P : \mathtt{Prop}}{\Gamma \vdash \exists x : T, P : \mathtt{Prop}} \quad (\mathrm{P}\text{-}\exists)$$

$$\frac{\Gamma \vdash e : T \qquad \Gamma \vdash P[e]}{\Gamma \vdash \exists x : T, P[x]} \quad (\exists\text{-}\mathrm{I})$$

$$\frac{\Gamma \vdash \exists x : T, P[x] \qquad \Gamma, x : T, H : P[x] \vdash Q \qquad \Gamma \vdash Q : \mathtt{Prop}}{\Gamma \vdash Q} \quad (\exists\text{-}\mathrm{E})$$

特称量化に関する証明(1)

```
Definition some_nat_is_even : Prop :=
  (* 偶数が存在する ∃ x : nat, ev x *)
  ex nat ev.
  (* Notation により
       exists x : nat, ev x
     でも OK *)

Definition snie : some_nat_is_even :=
  ex_intro _ ev
           4 (* 偶数をひとつ *)
           (ev_SS 2 (ev_SS 0 ev_0)).
             (* それが偶数である証拠 *)
```

特称量化に関する証明(2)

witness を指定して apply ex_intro を使う

```
Example exists_example_1 :
  exists n, n + (n * n) = 6.
Proof.
  apply ex_intro with (witness:=2).
  reflexivity.  Qed.
```

- apply ex_intro with (withness:=e) の代わりに exists e でも OK

特称量化に関する証明(3)

文脈に特称量化がある時は `inversion` **を使う**

- **H : ∃x, P[x] で** `inversion H` **とする**

  ⇒ **x と H′ : P[x] (x が P を満たすこと) が文脈へ**

```
Theorem exists_example_2 : forall n,
  (exists m, n = 4 + m) ->
  (exists o, n = 2 + o).
Proof.
  intros n H.
  inversion H as [m Hm].
    (* witness に intro パターンで名前をつける *)
  exists (2 + m).
  apply Hm.  Qed.
```

本日のメニュー

Logic.v (Coq の論理)

- 量化と含意
- 連言 (「かつ」)
- 選言 (「または」)
- 偽・矛盾
- 否定 (「〜でない」)
- 特称量化 (「ある x が存在して〜」)
- 等号
  - 再び inversion について
- 命題としての関係
- 非形式的証明

等号の定義(バージョン1)

```
Inductive eq (X:Type) : X -> X -> Prop :=
  refl_equal : forall x, eq X x x.
```

- **与えられた型** X**，その要素** x, y **について，** eq X x y **は命題**
- **等しさの証拠** refl_equal x: **「**x **と** x **は等しい」**
- Coq **は「計算により等しいもの」で置き換えた命題を同一視**

```
Definition four : eq _ (2 + 2) (1 + 3) :=
  refl_equal nat 4.
  (* four の型は eq _ 4 4 と同一視される *)
```

等号の定義(バージョン2)

Coq ライブラリ中での定義:

```
Inductive eq' (X:Type) (x:X) : X -> Prop :=
    refl_equal' : eq' X x x.
    (* forall が前に *)
```

- 論理的には等価
- ただし，帰納法の原理が違う

eq' の帰納法の原理

```
Check eq_ind'.
```

いわゆる「ライプニッツの同値性 (Leipnitz equality)」

ライプニッツの同値性

「x と y が等しい」とは x について成立する全ての性質 P が y でも成立することである

再び inversion タクティックについて

`inversion H` の挙動に関する統一的な説明:

- `H` の型が P で P は帰納的に定義 (等号の場合あり) されているとする
- P の各コンストラクタ C について
  - H が C からできていると仮定して得られるゴールを生成
  - C の引数 (規則の前提に相当するもの) を仮定として追加
  - C の型と P を比較して，成立すべき等式を計算して，
  - 文脈に追加 & 等式によるゴールの書き換え
  - もし等式が矛盾していたら，その時点でゴールは解消

例: 「または」に関する inversion

- P は Q ∨ R の形
- or_intro1, or_intror に対応するふたつのサブゴールが生成され，それぞれ，仮定に Q, R が追加される
- Q, R の形には制限がないので等式は特に生成されない

例: 「かつ」に関する inversion

- **P は Q ∧ R の形**
- **`conj` に対応してひとつのサブゴールが生成され，仮定に (同時に) Q, R が追加される**
- **Q, R の形には制限がないので等式は特に生成されない**

例: 等号に関する inversion

- P は $e_1 = e_2$ の形
- `refl_equal` に対応してサブゴールはひとつ
- 前提となる命題はない
- `refl_equal` の型が `eq X x x` なことにより,「e_1 と e_2 は等しい」から導かれる制約が文脈に追加

inversion タクティック (1) (再掲)

$$\begin{array}{l} \vdots \\ \mathbf{H} : \mathbf{c}\ \mathbf{a_1}\ \cdots\ \mathbf{a_n} = \mathbf{d}\ \mathbf{b_1}\ \cdots\ \mathbf{b_m} \\ \vdots \\ \hline \mathbf{P} \\ \quad \Downarrow \texttt{inversion H.}\ (\mathbf{c}, \mathbf{d}\ \text{が同じ場合}) \\ \vdots \\ \mathbf{H_1} : \mathbf{a_1} = \mathbf{b_1} \\ \vdots \\ \mathbf{H_n} : \mathbf{a_n} = \mathbf{b_n} \\ \vdots \\ \hline \mathbf{P'}\ (\mathbf{P}\ \text{に対して}\ \mathbf{H_1}, \ldots, \mathbf{H_n}\ \text{を使い書き換えた結果}) \end{array}$$

inversion (2) (再掲)

$$\frac{\begin{array}{l}\vdots \\ \mathbf{H} : \mathbf{c}\ \mathbf{a}_1\ \cdots\ \mathbf{a}_n = \mathbf{d}\ \mathbf{b}_1\ \cdots\ \mathbf{b}_m \\ \vdots\end{array}}{\mathbf{P}}$$

$\Downarrow$ inversion H. (**c**, **d** が**違う**場合)

仮定が矛盾しているので，このゴールは解消

例: 偶数性に関する inversion

- P は even e の形
- サブゴールはふたつ (ev_0 と ev_SS に対応)
- ev_0 に対応したサブゴールでは以下が文脈に追加
  - 「e と O は等しい」から導かれる制約
- ev_SS に対応したサブゴールでは以下が文脈に追加
  - 「n は偶数」(n:nat と H1:even n)
  - 「e と S(Sn) は等しい」から導かれる制約

本日のメニュー

Logic.v (Coq の論理)

- 量化と含意
- 連言 (「かつ」)
- 選言 (「または」)
- 偽・矛盾
- 否定 (「〜でない」)
- 特称量化 (「ある x が存在して〜」)
- 等号
- **命題としての関係**
- **非形式的証明**

命題としての関係

- **一引数の命題 (一引数述語): 「もの」の性質を表す**
  - `beautiful`, `even` **など**
- **二引数の命題 (二引数述語): 「もの」と「もの」の関係を表す**
  - `eq`

関係「以下」の帰納的定義(バージョン1)

```
Inductive le : nat -> nat -> Prop :=
  | le_n : forall n, le n n
  | le_S : forall n m, (le n m) -> (le n (S m))
```

導出規則:

$$\frac{}{\mathbf{n} \leq \mathbf{n}} \quad (\text{LE-N})$$

$$\frac{\mathbf{n} \leq \mathbf{m}}{\mathbf{n} \leq \mathrm{S}\ \mathbf{m}} \quad (\text{LE-S})$$

バージョン2

n がふたつのコンストラクタに共通しているので…

```
Inductive le (n:nat) : nat -> Prop :=
  | le_n : le n n
  | le_S : forall m, (le n m) -> (le n (S m)).
```

と定義してもよい．むしろこの方が帰納法が使いやすくなる．(前述した `eq` **の改良と同じ)**

```
Notation "m <= n" := (le m n).
```

「以下」に関する証明

基本的には even **などと同じ**:

- **ゴールにあるならコンストラクタを** apply
- **文脈にあるなら** inversion

```
Theorem test_le1 :  3 <= 3.
Proof.  apply le_n.  Qed.

Theorem test_le2 : 3 <= 6.
Proof.
  apply le_S. apply le_S.
  apply le_S. apply le_n.  Qed.
```

```
Theorem test_le3 : ~ (2 <= 1).
Proof.
  intros H.
  inversion H. inversion H1.  Qed.
```

「未満」の定義

```
Definition lt (n m:nat) := le (S n) m.
Notation "m < n" := (lt m n).
```

- `le` を使わずに直接帰納的な定義をするとしたら？

本日のメニュー

Logic.v (Coq の論理)

- 量化と含意
- 連言 (「かつ」)
- 選言 (「または」)
- 偽・矛盾
- 否定 (「〜でない」)
- 特称量化 (「ある x が存在して〜」)
- 等号
- 命題としての関係
- 非形式的証明
  - 非形式的な帰納法による証明
    - ★ 帰納的に定義された集合についての帰納法
    - ★ 帰納的に定義された命題についての帰納法

非形式的証明について

- 非形式的証明は形式的証明を再構成する方法を「教える」ものであるべき
- 非形式的証明の詳しさは時と場合による
  - 極端な場合: 形式的証明と同じくらい詳しく

$\Longrightarrow$ 形式的証明は再構成できるかもしれないが，何も「教えて」くれない

  - 逆の極端な場合: 「これは成立するから証明は頑張って考えろ.」

$\Longrightarrow$ 必要な洞察が多すぎて再構成できない

- ちょうどよい詳しさ:
  - 必要な洞察は全て与える
  - わかりきったところは適宜省略

非形式的な帰納法による証明

帰納法による証明のテンプレート:

- 帰納的に定義された集合 (型) についての帰納法
- 帰納的に定義された命題についての帰納法

帰納的に定義された集合 (型) についての帰納法

S を帰納的に定義された集合 (型) とする

定理: 任意の $n : S$ について，$P(n)$

証明: n についての帰納法による．
(S の各コンストラクタ c_i について)

- $n = c_1\ a_1\ \cdots\ a_k$ の場合: $P(c_1\ a_1\ \cdots\ a_k)$ を示す．. . . (型が S の各 a_j について帰納法の仮定 $P(a_j)$ を使う) . . .
- $n = c_2\ b_1\ \cdots\ b_m$ の場合: $P(c_2\ b_1\ \cdots\ b_m)$ を示す．. . . (型が S の各 b_j について帰納法の仮定 $P(b_j)$ を使う) . . .

定理: 任意の集合 X，リスト l : list X，数 n についてもし length l = n ならば，index (S n) l = None

証明: l についての帰納法による．

- l = [] の場合: 「任意の数 n について，もし length [] = n ならば，index (S n) [] = None」を示す．
  これは，index の定義より明らか．
- ある x, l′ について，l = x :: l′ かつ任意の length l′ = n′ なる n′ について index (S n′) l′ = None が成り立つ場合:
  「任意の数 n について，もし length (x :: l′) = n ならば，index (S n) (x :: l′) = None」を示す．

まず，n を length (x :: l′) = n なる自然数とする．

$$n = \mathrm{length}\ (x :: l') = S\ (\mathrm{length}\ l')$$

より，

$$\mathrm{index}\ (S\ (\mathrm{length}\ l'))\ l' = \mathrm{None}$$

を示せば十分．これは帰納法の仮定から (n′ として length l′ をとれば) 明らか．

帰納的に定義された命題についての帰納法

Q $x_1 \cdots x_n$ (Q $\vec{x}$ と書く) を帰納的に定義された命題とする

定理: 任意の $\vec{x}$ について，Q($\vec{x}$) ならば P($\vec{x}$)

証明: Q の導出についての帰納法による．
(Q の各コンストラクタ c_i について)

- Q $\vec{x}$ が規則 c_1 で導出された場合:
  - (ここで c_1 の結論の形からいえる x_i についての等式，c_1 への引数 (前提) の型，帰納法の仮定を述べる)
  - (示すべき P を述べる)
  - (P を示す)

定理: $\leq$ は推移的．すなわち，任意の数 n, m, o について，$n \leq m$ かつ $m \leq o$ ならば $n \leq o$．

証明: $m \leq o$ の導出に関する帰納法による．

- 最後の規則が `le_n` の場合．
  この時，$m = o$ であり，示すべきことは
  $n \leq o = m$ であるが，これは仮定より明らか．
- 最後の規則が `le_S` の場合．
  この時，ある o' が存在し，$o = S\ o'$ かつ，
  $m \leq o'$．この時，帰納法の仮定

  $$n \leq m \text{ かつ } m \leq o' \text{ ならば} n \leq o'$$

  を使って，$n \leq o = S\ o'$ を示す．
  帰納法の仮定より $n \leq o'$ がいえ，規則 `le_S` より
  $n \leq S\ o'$ がいえる．

宿題： 1/23 午前10:00 締切

- Exercise: `contrapositive` (2), `not_both_true_and_false` (1), `not_eq_beq_false` (2), `dist_not_exists` (1) `dist_exists_or` (2), `total_relation` (2)
- **解答を書き込んだ `Logic.v` をまるごとオンライン提出システムを通じて提出**
- **以下をコメント欄に明記**:
  - **講義・演習に関する質問，わかりにくいと感じたこと，その他気になること．(「特になし」はダメです．)**
  - **友達に教えてもらったら、その人の名前，他の資料 (web など) を参考にした場合，その情報源 (URL など)．**

1/22の講義について

- 黒板での演習
- 配布の演習問題を黒板で解く
- 成績への反映あり
- 問題の順番は問わない
- 早い物勝ち

期末試験について

- Coq のプログラム (型・命題定義，関数定義) は書ける必要あり
- Coq のタクティックを使った証明は書けなくてよい
- (日本語の) 非形式的証明は書ける必要あり
- 自然演繹，型付け，簡約の導出は書ける必要あり