「計算と論理」 Software Foundations その6

五十嵐 淳
cal14@fos.kuis.kyoto-u.ac.jp
igarashi@kuis.kyoto-u.ac.jp

京都大学

January 6, 2015

Logic.v

- 命題
- 証明とエビデンス
- 量化と含意
- 連言（「かつ」）
- 論理的同値 (if and only if)
- 選言（「または」）
- 偽・矛盾
- 否定（「〜でない」）

命題

Coq では，命題も (プログラムと同じく) 項の一種.
“Prop” に分類される.

```
Coq < Check (3=3).
3 = 3
     : Prop

Coq < Check (forall n:nat, n = 2).
forall n : nat, n = 2
     : Prop

Coq < Check (forall (n:nat) (b:bool), n = b).
Toplevel input, characters 36-37:
> Check (forall (n:nat) (b:bool), n = b).
>                                     ^
Error: In environment
n : nat
b : bool
```

証明とエビデンス

- 型が要素を持つのと同様に，命題も要素を持つ
- 命題に属する要素は「証明オブジェクト」

```
Coq < Lemma silly : 0 * 3 = 0.

Coq < Proof. reflexivity. Qed.

Coq < Print silly.
silly = eq_refl
     : 0 * 3 = 0
```

- eq_refl が証明オブジェクト
  - (今は，意味はわからなくてよいです．)

含意の証明オブジェクトは関数

```
Coq < Lemma silly_implication :
Coq <   (1 + 1) = 2 -> 0 * 3 = 0.

Coq < Proof. intros H. reflexivity. Qed.

Coq < Print silly_implication.
silly_implication =
fun _ : 1 + 1 = 2 => eq_refl
     : 1 + 1 = 2 -> 0 * 3 = 0
```

全称量化の証明オブジェクトも関数

```
Coq < Lemma silly_quantification :
Coq <   forall x:nat, 0 + x = x.

Coq < Proof. intros x. reflexivity. Qed.

Coq < Print silly_quantification.
silly_quantification =
fun x : nat => eq_refl
     : forall x : nat, 0 + x = x
Argument scope is [nat_scope]
```

Coq における命題の追加方法

型を追加するのと同じように `Inductive` を使う!

- 命題の追加方法
  1. 命題の名前を決める
  2. 命題が成立する条件 (命題に関する公理) を与える
     - ★ 公理 ≒ 証明オブジェクトのコンストラクタ!

c.f. 型の追加方法

1. 型の名前を決める
2. その型に属する値の作り方 (≒コンストラクタの型) を決める

Logic.v

- 命題
- 証明とエビデンス
- 量化と含意
- 連言 (「かつ」)
- 論理的同値 (if and only if)
- 選言 (「または」)
- 偽・矛盾
- 否定 (「～でない」)

連言 (conjunction)

「P かつ Q」の定義

```
Inductive and (P Q : Prop) : Prop :=
  conj : P -> Q -> (and P Q).
Notation "P /\ Q" := (and P Q) : type_scope.
```

- 命題をパラメータとする命題定義
- 直観: `and P Q` (`P` ∧ `Q`) の証拠は `P` の証拠と `Q` の証拠から (`conj` を付けることで) 構成される
  - `conj` が導入規則に相当している
- 逆に `P` ∧ `Q` の証拠があれば，そこから `P` の証拠と `Q` の証拠が取り出せる

直積の定義との比較

```
Inductive and (P Q : Prop) : Prop :=
  conj : P -> Q -> (and P Q).
```

```
Inductive prod (X Y : Type) : Type :=
  pair : X -> Y -> (prod X Y).
```

「かつ」の証明(1)

「かつ」に関する公理が「ならば」の形で述べられているので，今までのタクティクで証明できる

```
Theorem and_example :
  (0 = 0) /\ (3 = mult 2 2).
Proof.
  apply conj.   (* もしくは split. *)
  Case "left".  reflexivity.
  Case "right".  reflexivity.   Qed.

Print and_example.
```

「かつ」の証明 (2)

仮定に「かつ」が現れる場合:

```
Theorem proj1 : forall P Q : Prop,
  P /\ Q -> P.
Proof.
  intros P Q H.
  destruct H as [HP HQ].
  apply HP.  Qed.
```

- H は直感的には P の証明と Q の証明のペア $\Longrightarrow$ destruct で分解
  - 教科書では inversion で分解している

(論理的) 同値

同値 (if and only if) は，両方向の含意の連言:

```
Definition iff (P Q : Prop) :=
  (P -> Q) /\ (Q -> P).
Notation "P <-> Q" := (iff P Q)
  (at level 95, no associativity) : type_scope.
```

同値性に関する性質

```
Theorem iff_implies : forall P Q : Prop,
  (P <-> Q) -> P -> Q.
Proof.   (* 実は proj1 の特殊ケース *)



Qed.

Theorem iff_sym : forall P Q : Prop,
  (P <-> Q) -> (Q <-> P).
Proof.   (* 実は and_commut の特殊ケース *)



Qed.
```

Logic.v

- 命題
- 証明とエビデンス
- 量化と含意
- 連言(「かつ」)
- 論理的同値 (if and only if)
- 選言(「または」)
  - 「かつ」「または」と `andb`, `orb`
- 偽・矛盾
- 否定(「〜でない」)

選言 (disjunction)

「P または Q」の (帰納的な) 定義

```
Inductive or (P Q : Prop) : Prop :=
  | or_introl : P -> or P Q
  | or_intror : Q -> or P Q.
```

```
Notation "P \/ Q" := (or P Q) : type_scope.
```

- 直観—`or P Q` (`P` ∨ `Q`) の証拠を構成する方法は二通り:
  - `P` の証拠から構成
  - `Q` の証拠から構成

「または」についての証明(1)

```
Check or_introl.
Check or_intror.
```

「または」についての証明(2)

```
Theorem or_commut : forall P Q : Prop,
  P \/ Q -> Q \/ P.
Proof.
  intros P Q H.  destruct H as [HP | HQ].
    Case "left".  apply or_intror. apply HP.
    Case "right". apply or_introl. apply HQ.
Qed.
```

H は

- `or_introl` $P\ Q\ HP$ (ただし $HP : P$) の形か
- `or_intror` $P\ Q\ HQ$ (ただし $HQ : Q$) の形

のいずれか $\Longrightarrow$ `destruct` (または `inversion`) による場合分け

「または」についての証明(3)

```
Theorem or_commut' : forall P Q : Prop,
  P \/ Q -> Q \/ P.
Proof.
  intros P Q H.  destruct H as [HP | HQ].
    Case "left".  right. apply HP.
    Case "right". left.  apply HQ.
Qed.
```

- left. は apply or_introl. の略
- right. は apply or_intror. の略

「かつ」「または」と andb, orb

論理結合子 $\land$, $\lor$ と真偽値上の関数 andb, orb の関係

```
Theorem andb_true__and : forall b c,
  andb b c = true -> b = true /\ c = true.

Theorem and__andb_true : forall b c,
  b = true /\ c = true -> andb b c = true.

Theorem andb_false : forall b c,
  andb b c = false -> b = false \/ c = false.
```

など

Logic.v

- 命題
- 証明とエビデンス
- 量化と含意
- 連言 (「かつ」)
- 論理的同値 (if and only if)
- 選言 (「または」)
- 偽・矛盾
- 否定 (「～でない」)

偽 (falsehood)

「偽」(⊥ とも書かれる) の定義

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

- コンストラクタが存在しない定義!
- 偽の証明は存在しない
- 導入規則も存在しない

「偽」についての証明(1)

```
Check False_ind.

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

- destruct: 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文脈に矛盾を発生させること:

```
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.
```

Logic.v

- 命題
- 証明とエビデンス
- 量化と含意
- 連言 (「かつ」)
- 論理的同値 (if and only if)
- 選言 (「または」)
- 偽・矛盾
- 否定 (「〜でない」)
  - 不等号 (等しくない)

否定

「P ではない」($\neg P$, $\sim P$) 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) 少しコツが必要なことも.

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

Theorem contradiction_implies_anything :
  forall P Q : Prop, (P /\ ~P) -> Q.
Proof.
  intros P Q H. destruct 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.
```

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: $((P \text{ -> } Q) \text{ -> } P) \text{ -> } P$
- 排中律: $P \vee \neg P$
  - ただし，$\neg\neg(P \vee \neg P)$ は成立
- ド・モルガン則 (の一部): $\neg(\neg P \wedge \neg Q) \text{ -> } P \vee Q$
- $(P \text{ -> } Q) \text{ -> } (\neg P \vee Q)$

不等号

$x <> y$ は $\neg(x = y)$ 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.
```

宿題： 1/20 午前10:30 締切

- Exercise: `and_assoc` (2), `iff_properties` (1), `or_distributes_over_and_2` (2), `contrapositive` (2), `not_both_true_and_false` (1), `false_beq_nat` (2)
- 解答を書き込んだ `Logic.v` までのファイルを全てを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).