「計算と論理」
Software Foundations
その2

五十嵐 淳
igarashi@kuis.kyoto-u.ac.jp

京都大学

October 16, 2012

宿題について

- 先週の宿題の締切は 10/24 8:00 に延長しました
- (早めに) 登録願を出した人には提出システムについての連絡がいったと思います
- 自習・宿題のやり方について WWW に少し書きました

宿題コメントより (1)

simpl と reflexivity について: simpl を使わずとも reflexivity が式の単純化をしてくれるとのことですが、つまり reflexivity は simpl の動作を行なった後に両辺の等価性を確認するというコマンドと考えてもよろしいですか？

reflexivity. だけで証明を完了させるのに向いている場合と、simpl. などで簡約化した方が向いている場合の違いがよくわかりません。

回答

- **基本，よいです．**reflexivity **は**
  - simpl **だけでは見た目が等しくならない場合にも成功する場合があります．**
  - **等号ひとつからなる命題 (を全称量化したもの) にしか使えません．(そのうち，そうでない命題が出てきます．)**
- **(ちなみに，**Coq **では** `Definition` **や** `Check` **などのおまじないをコマンド，証明中に使うおまじないをタクティックと呼びわけています．)**

宿題コメントより (2)

- 「証明をするときに、ある程度省いても、証明ができていたので、どこまで詳しく書く必要があるのかが、分かりにくかったです。」
  - 大事な問題意識です．今日，そのあたりについて少し話します．
- 「intros タクティックの動作を理解するのに時間がかかりました。」
  - 「ならば」「任意の〜」は論理の最も基本かつ深い概念といってもよいと思います．intros を理解するのに時間がかかるのは，きちんと考えて取り組んでいる証拠だと思います．

今日のメニュー

Basics.v 後半

- **場合分けによる証明** (destruct **タクティック**)
- **場合分けに名前をつける** (Case **タクティック**)
- **数学的帰納法による証明** (induction **タクティック**)
- **形式的証明と非形式的証明**
- **証明中の証明** (assert **タクティック**)

場合分け: 単純化による証明の限界

変数を含む式は (最後まで) 計算できないことがある

```
Theorem plus_1_neq_0_firsttry : forall n : nat,
  beq_nat (n + 1) 0 = false.
Proof.
  intros n. simpl. (* does nothing! *)
```

場合分け: 単純化による証明の限界

変数を含む式は (最後まで) 計算できないことがある

```
Theorem plus_1_neq_0_firsttry : forall n : nat,
  beq_nat (n + 1) 0 = false.
Proof.
  intros n. simpl. (* does nothing! *)
```

- `+` は左側の数についての場合分けで定義されているので，`n + 1` の計算はこれ以上進まない
  - `S n` と `n + 1` の違い

場合分けによる証明

n が具体的にどんな形をしうるかを考えると計算が進む (場合がある)!

- (n = O の場合): n + 1 は単純化で 1 になる
- (n = S(…) の場合): n + 1 は単純化で S(S(…)) になる

いずれの場合も，+ はおろか，beq_nat の計算も完了する!

destruct タクティックによる場合分け

```
Theorem plus_1_neq_0 : forall n : nat,
  beq_nat (n + 1) 0 = false.
Proof.
  intros n. destruct n as [| n'].
    reflexivity. (* n = 0 の場合 *)
    reflexivity. (* n = S(…) の場合 *)
Qed.
```

- destruct 前後のゴールの変化に注目
- "…" 部分に名前をつける intro パターン
  - [] 内に，変数列を | で区切って並べる
  - 変数列の数 = 場合分けの数

今日のメニュー

Basics.v 後半

- 場合分けによる証明 (destruct タクティック)
- 場合分けに名前をつける (Case タクティック)
- 数学的帰納法による証明 (induction タクティック)
- 形式的証明と非形式的証明
- 証明中の証明 (assert タクティック)

場合分けに名前をつける

- 場合分けの証明は読みにくい
  - どこまでが「ひとつの場合」なの!?

⇒ コメント・インデントをつける？

  - 今どの場合を証明しようとしているか (特にコメントが画面の外に流れると) わかりにくい

⇒ `Case` タクティック！

  - この教科書の著者謹製
  - 教科書をこの部分まで読み込むと使えるようになる

Case を使って書き直した証明

```
Theorem plus_1_neq_0 : forall n : nat,
  beq_nat (n + 1) 0 = false.
Proof.
  intros n. destruct n as [| n'].
  Case "n = 0".
    reflexivity.
  Case "n = S n'".
    reflexivity.
Qed.
```

デモ: 単なるコメントとの違い

- andb_true_elim1 **の証明**

Case タクティック

- Case の使用は強制ではないが強く推奨
  - 後で読み返してわかる証明を書こう!
  - (Case に限らず，一行の長さとかも気をつけよう!)
- 場合分けが入れ子になる時のための SCase, SSCase, ...
  - SCase = subcase

今日のメニュー

Basics.v 後半

- 場合分けによる証明 (destruct タクティック)
- 場合分けに名前をつける (Case タクティック)
- **数学的帰納法による証明 (induction タクティック)**
- 形式的証明と非形式的証明
- 証明中の証明 (assert タクティック)

帰納法による証明

定理: 0 は足し算の右単位元

```
Theorem plus_O_r : forall n:nat,
  n + O = n.
```

詰まる証明

```
Proof.
  intros n. simpl. (* Does nothing! *)
```

「こういう時は場合分けでしょ？」

またもや詰まる証明

```
Proof.
  intros n. destruct n as [| n'].
  Case "n = 0".
    reflexivity. (* so far so good... *)
  Case "n = S n'".
```

- **場合分けをいくら続けてもキリがない!**
- **`n` より 1 小さい `n'` について `plus_O_r` が成り立っていれば…**

「こういう時は場合分けでしょ？」

またもや詰まる証明

```
Proof.
  intros n. destruct n as [| n'].
  Case "n = 0".
    reflexivity. (* so far so good... *)
  Case "n = S n'".
    simpl. (* また同じようなゴールが… orz *)
```

- **場合分けをいくら続けてもキリがない!**
- **n より 1 小さい n' について plus_0_r が成り立っていれば…**

「こういう時は場合分けでしょ？」

またもや詰まる証明

```
Proof.
  intros n. destruct n as [| n'].
  Case "n = 0".
    reflexivity. (* so far so good... *)
  Case "n = S n'".
    simpl. (* また同じようなゴールが… orz *)
```

- **場合分けをいくら続けてもキリがない!**
- **n より 1 小さい n' について plus_0_r が成り立っていれば… ⇒ 数学的帰納法**

数学的帰納法

P(n) を自然数 n の性質について述べた命題とする

数学的帰納法の原理

「任意の自然数 n について P(n)」は以下と同値

- P(0) かつ
- 任意の自然数 n' について P(n') ならば P(S n')

単なる場合分けと違って，P(S n') を示すのに，ひとつ小さい数では P が成立していること (つまり P(n')) を仮定してよい

- P(n') を「帰納法の仮定」(induction hypothesis, IH) と呼ぶ

数学的帰納法の妥当性

個々の具体的な数 (例えば 4) について P が成立することが,

- P(0) かつ
- 任意の自然数 n′ について P(n′) ならば P(S n′)

を組み合わせて導き出せる

数学的帰納法を使った証明

```
Theorem plus_O_r : forall n:nat, n + O = n.
Proof.
  intros n. induction n as [| n'].
  Case "n = O". reflexivity.
  Case "n = S n'".
    simpl. rewrite -> IHn'. reflexivity. Qed.
```

基本的な使い方は `destruct` **と同じ**

- intro **パターン**
- `IHn'` **が帰納法の仮定** (Coq **が勝手に名前をつける**)

今日のメニュー

Basics.v 後半

- 場合分けによる証明 (destruct タクティック)
- 場合分けに名前をつける (Case タクティック)
- 数学的帰納法による証明 (induction タクティック)
- **形式的証明と非形式的証明**
- 証明中の証明 (assert タクティック)

(数学的命題)の「証明」とは何か

- 読者に「主張が真であること」を納得してもらうための文章

$\Rightarrow$ 証明はコミュニケーション行為

(数学的命題)の「証明」とは何か

- 読者に「主張が真であること」を納得してもらうための文章

  $\Rightarrow$ 証明はコミュニケーション行為

- 「読者」が Coq の場合:
  - 証明は記号の羅列 (形式的)
  - 納得 = 証明検査アルゴリズムが yes を返す
    - 証明が，予め定まった規則に従って書かれているかの検査

(数学的命題)の「証明」とは何か

- 読者に「主張が真であること」を納得してもらうための文章

  $\Rightarrow$ 証明はコミュニケーション行為

- 「読者」が Coq の場合:
  - 証明は記号の羅列 (形式的)
  - 納得 = 証明検査アルゴリズムが yes を返す
    - ★ 証明が，予め定まった規則に従って書かれているかの検査
- 読者が人間の場合
  - 証明は自然言語で書かれる (非形式的)

(数学的命題)の「証明」とは何か

- 読者に「主張が真であること」を納得してもらうための文章

  $\Rightarrow$ 証明はコミュニケーション行為

- 「読者」が Coq の場合:
  - 証明は記号の羅列 (形式的)
  - 納得 = 証明検査アルゴリズムが yes を返す
    - ★ 証明が，予め定まった規則に従って書かれているかの検査
- 読者が人間の場合
  - 証明は自然言語で書かれる (非形式的)
  - 納得 = 書いてあることが信じられるか
    - ★ 人によって基準が違う!
    - ★ 人類が培ってきた数学・論理の流儀はある

形式的証明 vs. 非形式的証明

- 講義で主に扱うのは形式的証明
- でも，非形式的証明を軽視してはいけない
- 形式的証明は人間同士のコミュニケーションのメディアとしてはあまり効率的ではない

定理: + は結合的

```
Theorem plus_assoc' : forall n m p : nat,
  n + (m + p) = (n + m) + p.
Proof. intros n m p. induction n as [| n'].
  reflexivity. simpl. rewrite → IHn'.
  reflexivity. Qed.
```

読めない，でも，Coq にとっては正しい

きれいな非形式的証明

定理: 任意の n, m, p について $n + (m + p) = (n + m) + p$ である

証明: n についての帰納法.

- $n = 0$ とする.

$$0 + (m + p) = (0 + m) + p$$

を示す必要があるが，これは $+$ の定義より明らか.

- $n = S\ n'$ ただし，

$$n' + (m + p) = (n' + m) + p$$

とする．

$$(S\ n') + (m + p) = ((S\ n') + m) + p$$

を示す必要があるが，$+$ の定義より，これは

$$S(n' + (m + p)) = S((n' + m) + p)$$

と同値．これは帰納法の仮定より明らか．(証明終)

きれいな(構造がわかる)形式的証明

```
Theorem plus_assoc : forall n m p : nat,
  n + (m + p) = (n + m) + p.
Proof. intros n m p. induction n as [| n'].
  Case "n = 0".
    reflexivity.
  Case "n = S n'".
    simpl. rewrite → IHn'. reflexivity.
Qed.
```

- **非形式的証明との比較**:
  - **より明示的な部分**: simple, reflexivity
  - **明示的でない部分**: **途中のゴール**

今日のメニュー

Basics.v 後半

- 場合分けによる証明 (destruct タクティック)
- 場合分けに名前をつける (Case タクティック)
- 数学的帰納法による証明 (induction タクティック)
- 形式的証明と非形式的証明
- **証明中の証明 (assert タクティック)**

証明中の証明

- 以前に証明した定理は他の定理の証明中で使える
- 証明中でも「サブ定理」を宣言・証明できる

$\Rightarrow$ `assert` タクティック

```
Theorem mult_0_plus' : forall n m : nat,
  (0 + n) * m = n * m.
Proof.
  intros n m.
  rewrite → plus_0_n.


  reflexivity. Qed.
```

```
Theorem mult_0_plus' : forall n m : nat,
  (0 + n) * m = n * m.
Proof.
  intros n m.

  assert (H: 0 + n = n).
    Case "Proof of assertion". reflexivity.
  rewrite → H.
  reflexivity. Qed.
```

- `Case` は，読み易さのため
- `assert 0 + n = n as H` と書いてもよい

assert の挙動

- ゴールとして assert された命題が追加される
- 前のゴールの文脈には assert された命題が仮定として追加されている

assert の応用

そこじゃない!

```
Theorem plus_rearrange_firsttry :
  forall n m p q : nat,
    (n + m) + (p + q) = (m + n) + (p + q).
Proof.
  intros n m p q.
    (* n と m を入れ替えればいいんでしょ？ *)
  rewrite → plus_comm.
```

assert の応用

```
Theorem plus_rearrange : forall n m p q : nat,
  (n + m) + (p + q) = (m + n) + (p + q).
Proof.
  intros n m p q.
  assert (H: n + m = m + n).
    (* n と m の交換に特化 *)
    Case "Proof of assertion".
    rewrite -> plus_comm. reflexivity.
  rewrite -> H. reflexivity. Qed.
```

(こうしなきゃいけないのはどうかと思うが…)

宿題： 10/30 午前10:00 締切

- Exercise の zero_nbeq_plus_1, andb_true_elim2, basic_induction, plus_comm_informal, mult_comm
- **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，を自由に．(「特になし」はダメです．)**
- **解答を書き込んだ Basics.v をまるごとオンライン提出システムを通じて提出**
- **友達に教えてもらったら、その人の名前を明記**

来週は演習です

- 場所: 工学部 3 号館情報処理演習室 1
- 各自，宿題を進めてください
- 奥村君という ($\neq$ 初回に紹介した福田君)TA が来ます