「計算と論理」
Software Foundations
その2

五十嵐 淳
cal23@fos.kuis.kyoto-u.ac.jp

京都大学

October 10, 2023

別ファイルの定義・定理の読み込み

```
From LF Require Export Basics.
```

- LF という「論理フォルダ」から Basics の定義を読みこむ
  - _CoqProject というファイルで，LF がファイルシステム上のどこか指定している
- Basics.v を「コンパイル」した Basics.vo が必要
  - 最近は自動生成されるらしい (本当かな〜)
  - Induction.v 冒頭の説明も読んでください

`Induction.v`

- 数学的帰納法による証明 (`induction` タクティック)
- 証明中の証明 (`assert` タクティック)
- 形式的証明と非形式的証明

Induction.v

- 数学的帰納法による証明 (induction タクティック)
- 証明中の証明 (assert タクティック)
- 形式的証明と非形式的証明

帰納法による証明

定理: 0 は足し算の右単位元

```
Theorem add_0_r_firsttry : forall n:nat,
  n = n + 0.
```

詰まる証明

```
Proof.
  intros n. reflexivity. (* エラー! *)
```

何が起こっているのか？

```
Proof.
  intros n. simpl. (* 右辺が計算されない *)
```

「こういう時は場合分けだったよね？」

またもや詰まる証明

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

「こういう時は場合分けだったよね？」

またもや詰まる証明

```
Proof.
  intros n. destruct n as [| n'].
  - (* n = 0 *)
    reflexivity. (* so far so good... *)
  - (* 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 add_0_r が成り立っていれば…

「こういう時は場合分けだったよね？」

またもや詰まる証明

```
Proof.
  intros n. destruct n as [| n'].
  - (* n = 0 *)
    reflexivity. (* so far so good... *)
  - (* 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 add_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,

- $P(\mathbf{0})$ 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

1. $P(\mathbf{0})$ ならば $P(\mathbf{1})$
2. $\vdots$
3. $P(\mathbf{3})$ ならば $P(\mathbf{4})$

数学的帰納法を使った証明

```
Theorem add_0_r : forall n:nat, n = n + 0.
Proof.
  intros n. induction n as [| n' IHn'].
  - (* n = 0 *)
    reflexivity.
  - (* 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 `[| n' IHn' ]`
- `IHn'` ...帰納法の仮定につける名前

日本語で書くなら…

定理: 任意の自然数 n について $n + 0 = n$.

n についての数学的帰納法による.

- $n = 0$ の場合，示すべきは $0 + 0 = 0$ だが，これは $+$ の定義により自明.
- $n = S(n')$ の場合，示すべきは $S(n') + 0 = S(n')$ であるが,

$$
\begin{array}{rcll}
\text{左辺} & = & S(n' + 0) & + \text{ の定義による} \\
 & = & S(n') & \text{帰納法の仮定による} \\
 & = & \text{右辺} &
\end{array}
$$

なので証明終.

数学的帰納法を使った証明 (2)

```
Theorem minus_n_n : forall n,
  minus n n = 0.
```

今日のメニュー

`Induction.v`

- 数学的帰納法による証明 (induction タクティック)
- 証明中の証明 (assert タクティック)
- 形式的証明と非形式的証明

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」(補題) を宣言・証明できる

$\Rightarrow$ `assert` タクティック

assert を使った(やや人工的な)例

```
Theorem mult_0_plus' : forall n m : nat,
  (n + 0 + 0) * m = n * m.
Proof.
  intros n m.
  assert (H: n + 0 + 0 = n).
    { rewrite add_comm. simpl.
      rewrite add_comm. reflexivity. }
  rewrite -> H.
  reflexivity. Qed.
```

assert を使った(やや人工的な)例

```
Theorem mult_0_plus' : forall n m : nat,
  (n + 0 + 0) * m = n * m.
Proof.
  intros n m.
  assert (H: n + 0 + 0 = n).
    { rewrite add_comm. simpl.
      rewrite add_comm. reflexivity. }
  rewrite -> H.
  reflexivity. Qed.
```

- `assert (0 + n = n) as H` と書いてもよい

assert の挙動

- 新たなサブゴールとして `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 -> add_comm.
    (* ゴールが…思ってたのと違う! *)
```

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).
    (* この文脈での n と m の交換に特化 *)
    { rewrite -> add_comm. reflexivity. }
  rewrite -> H. reflexivity. Qed.
```

こんなことしなきゃいけないのはツールとしてどうかと思うが…

assert 使用上の注意

- (慣れるまでは) トップレベルで `Theorem` を使って証明しましょう
- 特に帰納法を使う必要がある補題を `assert` で証明しようとするとよく嵌まります
- 与えられた文脈で証明する必要性が高く，内容的にはほぼ自明なものに限りましょう

おまけ: replace タクティック

`replace (`A`) with (`B`)`: ゴール中の式 A を B で置き換える．$A = B$ は後で証明する (義務が生じる).

```
Theorem plus_rearrange' : forall n m p q : nat,
  (n + m) + (p + q) = (m + n) + (p + q).
Proof.
  intros n m p q.
  replace (n + m) with (m + n). (* 要括弧 *)
  (* ここでゴールの n + m が m + n になる *)
  - reflexivity.
  - (* n + m = m + n の証明 *)
    rewrite add_comm. reflexivity.
  Qed.
```

今日のメニュー

`Induction.v`

- 数学的帰納法による証明 (`induction` タクティック)
- 証明中の証明 (`assert` タクティック)
- 形式的証明と非形式的証明

(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 add_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．(すなわち，帰納法の $P(n)$ を m, p を仮定した上で $n + (m + p) = (n + m) + p$ とする．)

- $n = 0$ 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,

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

(すなわち $P(n')$) が成立していると仮定する.

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

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'].
  - (* n = O *) reflexivity.
  - (* 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: simpl, reflexivity
  - 明示的でない部分: 途中のゴール

宿題： / 午前10:30 締切

- `Induction.v` の `basic_induction` (2), `double_plus` (2), `add_comm_informal` (2), `eqb_refl` (2)
- `Induction.v` までのその他の問題は随意課題 (加点あり)
- 解答が記入された `Induction.v` を origin/master に push