「計算と論理」
Software Foundations
その6

五十嵐 淳
igarashi@kuis.kyoto-u.ac.jp

京都大学

November 27, 2012

コメント欄より

Q: `apply` の説明で出てくる具体化がよくわからない
A: 具体化は「任意の n について〜」という命題の具体例を得るプロセス

- 例: $\forall n\ m, n + m = m + n$ を具体化して (例えば)
  - $0 + 1 = 1 + 0$
  - $1 + 3 = 3 + 1$
  - などなど

  が得られる.

`apply` は，(一般的な性質を述べた) 定理の具体例を通じて，今のゴールを導く十分条件に遡るためのタクティック

Q: 最後の問題が難しい

```
Theorem plus_n_n_injective : forall n m,
      n + n = m + m ->
      n = m.
```

ヒント:

- **n, m がゼロかどうかで 4 通りの場合を考える**
- **うちふたつの場合は考えなくてもよい (ありえない) 場合**
- `induction` **を使うタイミングに注意**
- `apply in` **を使う練習だが，別に使わなくても解ける．(使わずに解けてしまったら，**`apply in`** を使う別解も考えてみよう．)**

今日のメニュー

- Poly.v **後半**: **もっと Coq について**
  - **複合的な式に関する** `destruct` **の使用**
  - `remember` **タクティック**
  - `apply ... with ...` **タクティック**
- Gen.v: **帰納法による証明について**
  - `generalize dependent` **タクティック**

複合的な式に関する場合分け

- destruct の引数は文脈にある変数でなくてもよい
  - (実は他のタクティックも変数以外の引数が取れる)
- 式一般についての場合わけが可能

```
Definition sillyfun (n : nat) : bool :=
  if beq_nat n 3 then false
  else if beq_nat n 5 then false
  else false.

Theorem sillyfun_false : forall (n : nat),
  sillyfun n = false.
```

- beq_nat の結果 (n が 3 かどうか，5 かどうか) についての場合分け
  - n が 5 以下の場合を個別に，$n \geq 6$ の場合を帰納法で証明，という手もあるかもしれないが…

```
Proof.
  intros n. unfold sillyfun.
  destruct (beq_nat n 3).
    Case "beq_nat n 3 = true". reflexivity.
    Case "beq_nat n 3 = false".
      destruct (beq_nat n 5).
      SCase "beq_nat n 5 = true". reflexivity.
      SCase "beq_nat n 5 = false". reflexivity.
Qed.
```

別の例

```
Definition sillyfun1 (n : nat) : bool :=
  if beq_nat n 3 then true
  else if beq_nat n 5 then true
  else false.

Theorem sillyfun1_odd : forall (n : nat),
     sillyfun1 n = true ->
     oddb n = true.
```

- `sillyfun1` **が真を返すための必要条件は「引数が奇数」**

```
Proof.
  intros n eq. unfold sillyfun1 in eq.
  (* eq : (if beq_nat n 3 then ...) = true
     ============================
     oddb n = true                          *)
  destruct (beq_nat n 3).
  Case "beq_nat n 3 = true".
  (* eq : true = true   ←左辺の単純化の結果
     ============================
     oddb n = true                          *)
```

- `beq_nat n 3 = true` の場合と，`beq_nat n 3 = false` の場合で分けたつもりなのに，肝心の `beq_nat n 3 = true` が消えている!!

remember タクティック

新しい変数とそれが何かと等しいことを文脈に記録

```
Proof.
  intros n eq. unfold sillyfun1 in eq.
  (* eq : (if beq_nat n 3 then ...) = true
     ============================
     oddb n = true                          *)
  remember (beq_nat n 3) as e3.
  (* eq : ...
     e3 : bool
     Heqe3 : e3 = beq_nat n 3
     ============================
     oddb n = true                          *)
```

ここで e3 についての場合分けをする

```
destruct e3.
Case "e3 = true".
(* eq : true = true
   e3 : bool   ← 消えているはず
   Heqe3 : true = beq_nat n 3
   ============================
   oddb n = true                    *)
```

あとは，Heqe3 から n = 3 がわかる．

```
apply beq_nat_eq in Heqe3.
(* Heqe3 : n = 3 になる *)
rewrite -> Heqe3.   (* n = 3 を代入 *)
reflexivity.
```

remember を使った証明のポイント

- destruct の場合分けは対象の式を直接書き換えてしまうので，**その式が含んでいた変数との関係性が失なわれる**ことがある
- 「〜と等しい変数 x」 を remember で導入して
- x について destruct による場合分けをする
- 元の式は書き換え対象にならないので関係性も残る!

今日のメニュー

- Poly.v 後半: もっと Coq について
  - 複合的な式に関する `destruct` の使用
  - `remember` タクティック
  - `apply ... with ...` タクティック
- Gen.v: 帰納法による証明について
  - `generalize dependent` タクティック

apply withタクティック

動機付け: 等号の推移律

```
Theorem trans_eq : forall X:Type (n m o : X),
  n = m -> m = o -> n = o.
```

をより具体的な例についての証明で使うことを考える.

```
Example trans_eq_example' :
  forall (a b c d e f : nat),
     [a,b] = [c,d] ->
     [c,d] = [e,f] ->
     [a,b] = [e,f].
Proof.
  intros a b c d e f eq1 eq2.
  apply trans_eq.    (* エラー! *)
```

何が起こったのか？

- **ゴール**: `[a,b] = [e,f]`
- `trans_eq :`
  `forall X n m o, n = m -> m = o -> n = o`

- Coq **がわかってくれること**:
  - `X` := `list nat`
  - `n` := `[a,b]`
  - `o` := `[e,f]`
- `m` **を何にすべきかはわからない**!

Coq にヒントを与える with

```
Example trans_eq_example' :
  forall (a b c d e f : nat),
      [a,b] = [c,d] ->
      [c,d] = [e,f] ->
      [a,b] = [e,f].
Proof.
  intros a b c d e f eq1 eq2.
  apply trans_eq with (m:=[c,d]).
  apply eq1. apply eq2.
Qed.
```

今日のメニュー

- Poly.v 後半: もっと Coq について
  - 複合的な式に関する `destruct` の使用
  - `remember` タクティック
  - `apply ... with ...` タクティック
- Gen.v: 帰納法による証明について
  - `generalize dependent` タクティック

帰納法による証明について

前回より:

- 帰納法は「任意の x について，P(x)」の形の判断を証明する技法
- P の選び方によって証明がうまくいったりいかなかったりする

```
Theorem double_injective : forall n m,
     double n = double m -> n = m.
```

- **うまくいく証明(の出だし)**

  `Proof. intros n. induction n as [| n'].`

  帰納法の P(n) は

  $$\forall \mathbf{m}, \mathbf{double\ n} = \mathbf{double\ m} \rightarrow \mathbf{n} = \mathbf{m}$$

- **うまくいかない証明(の出だし)**

  `Proof. intros n m. induction n as [| n'].`

  帰納法の P(n) は

  $$\mathbf{double\ n} = \mathbf{double\ m} \rightarrow \mathbf{n} = \mathbf{m}$$

  特定の m についての性質を証明することになってしまう.

m についての帰納法で証明してみる

```
Theorem double_injective : forall n m,
     double n = double m -> n = m.
Proof. intros n m. induction m as [| m'].
```

- これはうまくいかないパターン: 帰納法の P(m) は

$$\mathbf{double\ n = double\ m \rightarrow n = m}$$

  特定の n についての性質を証明することに…
- 全称量化の順番からいって先に n を文脈にいれないといけない
- 定理の文言を `forall m n,` とする？
  - 証明のやり方のせいで，定理の述べ方を変えなければいけない!?

generalize dependent タクティック

文脈で仮定した変数を再び全称量化するタクティック

```
Theorem double_injective_take2 : forall n m,
  double n = double m -> n = m.
Proof.
  intros n m.
  generalize dependent n.
  (* ゴールが望む forall n, ... の形になる *)
  induction m as [| m'].
  ...
```

仮定も道連れ

```
Theorem gen_example : forall n m,
  n = m -> S n = S m.
Proof.
  intros n m H.
  generalize dependent n.
(* ゴールは forall n, n = m -> S n = S m
   n に関する仮定が「ならば」の形でゴールに移る *)
```

宿題： 12/5 午前10:00 締切

- Exercise: `override_shadow` (1), `override_same` (2), `apply_exercises` (3), `gen_dep_practice` (3), `gen_dep_practice_opt` (3)
- **解答を書き込んだ `Poly.v` と `Gen.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)．**