Coqゼミ　第2回

浅井研究室（担当：廣田知子）

2008年6月23日

1　命題論理

命題変数　P, Q, ...
implication「ならば」　→

implication以外にも、論理関係としてandやor、notがあるはずですが、今回は→のみを考えていきます。（ちなみに→は右結合です。）

命題論理式の例：

$$(P \to Q) \to (Q \to R) \to (P \to R)$$

命題論理式の恒真性は、数理基礎論的には命題変数に可能な真偽値を割り当てて、全体の真偽値を調べます。

2　証明：命題：ソート

proof term : proposition : Prop

proposition
　型（ソート）がPropであるような式。
　例： P→P

proof term
　型がpropositionであるような式。
　例： fun (n : P) => n

3　命題変数の宣言

局所変数の宣言と全く同じ。型がPropになるだけ。

Variable 命題変数名 : Prop.
Variables 命題変数名 : Prop.

例：

```
Section propositional_logic.
  Variables P Q R T : Prop.
  ...
```

（これで以下、P, Q, R, Tという命題変数が使えるようになります。これらを使える場所として、ブロックを始めに宣言しています。）

4　大域的な補題・定理の定義

大域的変数定義(Definition)に近い。

Lemma 補題名 : 論理式.
Theorem 定理名 : 論理式.

補題と定理の差は感覚的なもの。重要なものは定理、補助的なものは補題。

補題、定理を定義すると、それが正しいことの証明をつけなくてはならない。（大域的変数定義時に変数の本体を与えなくてはならないのと同じ。）

5　証明の仕方

Proof.
　証明のための各種のコマンド
　...
Qed.

Goal　証明する命題のこと。
tactic　証明に使う各種コマンドのこと。

6　基本のtactic

intro 名前.

Goalが$A \to B$のとき、Aに名前を与えて、それを仮定に組み込む。GoalはBになる。一度に何回もintroをするときには**intros**を使う。名前を省略すると、システムが適当に名前を付けてくれる。（本格的な証明を自分で書く場合には、自分で名前をきちんと付けないと証明を読めなくなる。）

assumption.

Goalが仮定にあるときに使う。それで現在のGoalは証明終了となる。

apply 名前.

Goalに対して、名前で指定した仮定を使う。GoalがBで、使う仮定が$A \to B$のとき、新しいGoalはAになる。GoalがC

で、使う仮定が $A \to B \to C$ のとき、新しい Goal は A と B の二つになる。

7 exact コマンド

証明終了後、証明した定理・補題の名前を Print してみると、定理・補題に対応した関数が作られていることが分かる。実は、この式をそのまま与えると、それでも証明になる。つまり、

Proof.
 exact (定理・補題に対応した関数).
Qed.

とすれば、これでも証明になるということ。又、証明の途中で現在の Goal に対応する関数を exact コマンドで与えることも出来る。

8 Proof Irrelevance

Definition を使った変数定義と Theorem, Lemma を使った定理・補題は、どちらも「似たようなもの」である。が、前者は型よりも計算に意味があるのに対し、後者は計算よりも型（論理式）に意味がある。

9 Tactic の合成

セミコロン ; で二つのコマンドをつなぐと順に実行する。(ただし、複数の Goal がある場合には、全ての Goal に対して、そのコマンドが実行出来なくてはならない。)

[...|...] でどちらかのコマンドを実行する。二つ以上、並べても良い。

以上を任意にネストさせても良い。

練習問題

問 1 以下を証明せよ

```
(1) Lemma id_p : P -> P.
(2) Lemma id_pp : (P -> P) -> (P -> P).
(3) Lemma imp_trans : (P -> Q) -> (Q -> R) -> P -> R.
(4) Lemma imp_perm : (P -> Q -> R) -> (Q -> P -> R).
(5) Lemma diamond : (P -> Q) -> (P -> R) -> (Q -> R -> T) -> P -> T.
(6) Lemma weak_peirce : ((((P -> Q) -> P) -> P) -> Q) -> Q.
```