Coq チュートリアル
ウィンタースクール「数学ソフトウェア・チュートリアル」

溝口佳寛

九州大学マス・フォア・インダストリ研究所

http://imi.kyushu-u.ac.jp/~ym/

2015 年 2 月 18 日 (水)

本スライド : http://www.slideshare.net/yoshihiromizoguchi/20150218tutorial-44796812

サンプルファイル : https://github.com/KyushuUniversityMathematics/CoqExamples/tree/master/20150218

Institute of Mathematics for Industry
Kyushu University

目次

Institute of Mathematics for Industry
Kyushu University

数学定理の形式証明 (論文)

- G. Gonthier, Formal Proof—The Four-Color Theorem. Notices of the American Mathematical Society, 55(11), 13821393, 2008. `http://www.ams.org/notices/200811/tx081101382p.pdf`
- R. Affeldt and M. Hagiwara, Formalization of Shannon's Therems in SSReflect-Coq, Proc. 3rd Conference on Interactive Theorem Proving, LNCS 7406, 233249, 2012.
- G. Gonthier, et al., A Machine-Checked Proof of the Odd Order Theorem, Proc. 4th Conference on Interactive Theorem Proving, LNCS 7998, 163179, 2013. `https://hal.inria.fr/hal-00816699/file/main.pdf`
- F. Chyzak, A. Mahboubi et.al, A Computer-Algebra-Based Formal Proof of the Irrationality of $\zeta(3)$, Proc. 5th International Conference on Interactive Theorem Proving, LNCS 8558, 2014, `https://hal.inria.fr/hal-00984057`.
- T. Hales, Dense Sphere Packings : A blueprint for formal proofs, Cambridge University Press, 2012. (The Kepler Conjecture)
- ……

Institute of Mathematics for Industry
Kyushu University

数学定理の形式証明 (リンク, 広報, プロジェクト)

- SSreflect in the world,
  `http://coqfinitgroup.gforge.inria.fr/ssreflect_world.html`
- Coq Proof of the Four Color Theorem, 2006/04/26,
  `http://bit.ly/FourColorTheorem`
- Feit thompson proved in Coq, 2012/09/20,
  `http://bit.ly/FeitThompson`
- The announcement of the completion of the Flyspec project, 2014/8/10.
  `http://bit.ly/Flyspeck`
  (The Kepler Conjecture)
- Univalent Foundations of Mathematics, 2012,2013.
  `http://bit.ly/UnivalentFoundations`
  (Homotopy Type Theory)
- Computing close approximations of Pi,
  `http://www-sop.inria.fr/members/Yves.Bertot/proofs.html`

Institute of Mathematics for Industry
Kyushu University

証明支援系 Coq を使うには

本スライドの例題は Coq8.4pl5, ssreflect-1.5rc1, mathcomp-1.5rc1 で確認されています. https://github.com/KyushuUniversityMathematics/CoqExamples/tree/master/20150218
現在, Coq8.5beta1, ssreflect-1.5.coq85beta1, mathcomp-1.5.coq85beta1 が公表されています. https://coq.inria.fr/coq-85

- Mathlibre (Linux)
  ライブ DVD に, coq8.4pl3, ssreflect1.5, mathcomp1.5 が入っています.

  http://mirror.math.kyushu-u.ac.jp/mathlibre/mathlibre-debian-amd64-20141110-ja.iso

- Windows
  Coq8.5beta1,ssr-mathcomp-1.5 のインストーラがあります.

  https://coq.inria.fr/distrib/V8.5beta1/files/coq-installer-8.5beta1.exe

  http://ssr.msr-inria.inria.fr/FTP/ssr-mathcomp-installer-1.5coq8.5beta1.exe

- MacOSX
  Coq8.5,ssr-mathcomp-1.5 のインストーラがありますが確認していません.

  https://coq.inria.fr/coq-85

開発環境は CoqIDE, または, Emacs 上の ProofGeneral を使います. Coq8.5からは, Coq/jEdit も使えるらしいですが確認出来ていません.

Institute of Mathematics for Industry
Kyushu University

数学の定理 (1)

Let $\mathbf{N} = \{0, 1, \cdots\}$ be the set of natural numbers.

Theorem

For any natural number $n \in \mathbf{N}$, we have

$$\sum_{k=0}^{n} k = \frac{n(n+1)}{2}.$$

Where

$$\begin{cases} \sum_{k=0}^{0} k & = & 0, \\ \sum_{k=0}^{n+1} k & = & (n+1) + \sum_{k=0}^{n} k \qquad (n \in \mathbf{N}). \end{cases}$$

Institute of Mathematics for Industry
Kyushu University

関数 (プログラム) の定義

2 つの関数 $f : \mathbf{N} \to \mathbf{N}$, $g : \mathbf{N} \to \mathbf{N}$ を以下のように定義する.

1.
$$\begin{cases} f(0) & = & 0 \\ f(n+1) & = & (n+1) + f(n) \end{cases}$$

2.
$$g(n) = \frac{n(n+1)}{2}$$

但し, $n \in \mathbf{N}$ とします.

Institute of Mathematics for Industry
Kyushu University

数学の定理 (2)

先の定理は関数 f, g を用いて, 次のように書き換えられます.

Theorem

For any natural number $n \in \mathbf{N}$, we have

$$f(n) = g(n).$$

Institute of Mathematics for Industry
Kyushu University

数学の証明

(i) $n = 0$ のとき

$$f(0) = 0 = \frac{0(0+1)}{2} = g(0).$$

(ii) $f(n) = g(n)\left(= \frac{n(n+1)}{2}\right)$ を仮定するとき

$$\begin{aligned} f(n+1) &= (n+1) + f(n) \\ &= (n+1) + \frac{n(n+1)}{2} \\ &= \frac{(n+1)((n+1)+1)}{2} \\ &= g(n+1) \end{aligned}$$

(i),(ii) により数学的帰納法により全ての $n \in \mathbf{N}$ に対して,
$f(n) = g(n)$ が成立する.

Institute of Mathematics for Industry
Kyushu University

数学の補題 (1)

先の証明には次の補題が必要です.

Lemma

For any natural number $n \in \mathbf{N}$, we have

$$(n+1) + \frac{n(n+1)}{2} = \frac{(n+1)((n+1)+1)}{2}.$$

この証明は演算 + や /2 の帰納的な定義に従い証明されます.

Institute of Mathematics for Industry
Kyushu University

自然数と加算の帰納的定義

Definition

- $\mathbf{0} \in \mathbf{N}$,
- $p \in \mathbf{N} \to (S\ p) \in \mathbf{N}$.

Definition

- $\mathbf{0} + m = m$,
- $(S\ p) + m = (S\ (p + m))$.

一般に関数適用 $S(p)$ と書きますが, ここでは, Coq の記法に従い, $(S\ p)$ と書いています. さらに, $(S\ p)$ を $p.+1$ と書きます. 加算の + と同じ記号を含みますが注意して区別します. すなわち, $.+1 : \mathbf{N} \to \mathbf{N}$ に対し, $+ : \mathbf{N} \to (\mathbf{N} \to \mathbf{N})$ です. (注. Coq では, $(n + m)$ は $+(n)(m)$ を意味します.)

$$(p.+1) + m = (p + m).+1$$

Institute of Mathematics for Industry
Kyushu University

数学の補題 (2)

次の簡単な式も定義に従った数学的帰納法での証明が必要です.

Lemma

For any natural number $n \in \mathbf{N}$, we have

$$n + 0 = n$$

(i) $n = 0$ のとき

$$0 + 0 = 0.$$

(ii) $n + 0 = n$ を仮定するとき

$$\begin{aligned} (n. + 1) + 0 &= (n + 0). + 1 \\ &= n. + 1. \end{aligned}$$

(i),(ii) により数学的帰納法により全ての $n \in \mathbf{N}$ に対して, $n + 0 = n$ が成立する.

Institute of Mathematics for Industry
Kyushu University

Coq の定理と証明 (1)

Coq による簡単な定理 (補題) とその証明を紹介します.

```
Lemma lemma0: forall n:nat, n + 0 = n.
Proof.
  elim.                          (* n に対する数学的帰納法で証明 *)
  rewrite /addn/addn_rec/plus.
  apply (erefl 0).                    (* n=0 のときは明らか *)
  move => n H.
  rewrite /addn/addn_rec/plus.
  fold plus.
  rewrite plusE.
  rewrite H.                          (* n のときの仮定を使って *)
  apply (erefl (n.+1)).               (* n+1 のときの証明が出来る *)
Qed.
```

※ Proof. と Qed. の間は証明ではなく証明を作るためのコマンド列です!

Institute of Mathematics for Industry
Kyushu University

Coq の定理と証明 (2)

証明を作るためのコマンドで作成された証明そのものは
命題の型を持つ関数 (プログラム) です.

```
lemma0 =
fun _top_assumption_ : nat =>
(fun _evar_0_ : (fun n : nat => n + 0 = n) 0 =>
 (nat_ind (fun n : nat => n + 0 = n) _evar_0_)^~ _top_assumption_)
 (erefl 0)
 (fun (n : nat) (H : n + 0 = n) =>
  (fun _evar_0_ : (n + 0).+1 = n.+1 =>
   eq_ind_r
    (fun _pattern_value_ : nat -> nat -> nat =>
     (_pattern_value_ n 0).+1 = n.+1) _evar_0_ plusE)
   ((fun _evar_0_ : n.+1 = n.+1 =>
     eq_ind_r (fun _pattern_value_ : nat => _pattern_value_.+1 = n.+1)
       _evar_0_ H) (erefl n.+1)))
   : forall n : nat, n + 0 = n
```

Institute of Mathematics for Industry
Kyushu University

Coq の補題 (1)

既に証明された Ssreflect.ssrnat 中にある補題たち

```
Lemma add0n: forall n:nat, 0 + n = n.
Lemma addn0: forall n:nat, n + 0 = n.
Lemma addSn: forall m n:nat, m.+1 + n = (m + n).+1.
Lemma addnS: forall m n:nat, m + n.+1 = (m + n).+1.
Lemma addnC: forall n m:nat, m + n = n + m.
```

```
Lemma muln_divA d m n : d %| n -> m * (n %/ d) = m * n %/ d.
Lemma dvdnn m : m %| m.
Lemma muln1 n: n * 1 = n.
Lemma dvdn_mull d m n : d %| n -> d %| m * n.
Lemma divnDl m n d : d %| m -> (m + n) %/ d = m %/ d + n %/ d.
```

※ http://ssr.msr-inria.inria.fr/doc/ssreflect-1.5/Ssreflect.ssrnat.html

※ http://ssr.msr-inria.inria.fr/doc/mathcomp-1.5/MathComp.div.html

Institute of Mathematics for Industry
Kyushu University

Coq の補題 (2)

一旦, 自然数上で成立する演算の性質がわかると, その性質の補題を用いれば帰納法を用いることなく複雑な証明を構成することが出来ます.
また, 数式を簡潔な方向に変換して適用する補題を自動で探して証明する手続きがあります (ring など).
このような手続きを Tactic と呼びます. 以下は ring の利用例.

```
Lemma lemma1: forall n:nat,
  (n.+1)*2 + (n * (n.+1)) = (n.+1 * (n.+2)).
Proof.
  move => n.
  ring.
Qed.
```

Tactic を自分で作ることも出来ます.

Institute of Mathematics for Industry
Kyushu University

Coq の補題 (3)

```
Lemma lemma2: forall n:nat,
  (n.+1 * 2) %/ 2 = n.+1.
Proof.
  move => n.
  rewrite -(muln_divA (n.+1)).
  rewrite (divnn 2).
  simpl.
  by [rewrite muln1].
  apply (dvdnn 2).
Qed.

Lemma lemma3: forall n:nat,
  2 %| (n.+1 * 2).
Proof.
  move => n.
  apply dvdn_mull.
  apply (dvdnn 2).
Qed.
```

Institute of Mathematics for Industry
Kyushu University

Coq の補題 (4)

Lemma

For any natural number $n \in \mathbf{N}$, we have

$$(n+1) + \frac{n(n+1)}{2} = \frac{(n+1)((n+1)+1)}{2}.$$

```
Lemma lemma4: forall n:nat,
  n.+1 + (n * n.+1) %/ 2 = (n.+1 * n.+2) %/ 2.
Proof.
  move => n.
  rewrite -lemma1.
  rewrite (divnDl (n*n.+1) (lemma3 n)).
  by [rewrite lemma2].
Qed.
```

Institute of Mathematics for Industry
Kyushu University

Coq による関数実装と定理 (1)

2 つの関数 f, g は, Coq では以下のように実装出来ます.

```
Fixpoint f (n:nat) :=
match n with
| O => 0
| p.+1 => (p.+1) + (f p)
end.
```

```
Definition g (n:nat) := ((n * (n.+1)) %/ 2).
```

先の定理は, Coq では以下のように記述されます.

```
Theorem sigma: forall n:nat, (f n) = (g n).
```

Institute of Mathematics for Industry
Kyushu University

Coq による関数実装と定理 (2)

Theorem

For any natural number $n \in \mathbb{N}$, we have

$$f(n) = g(n).$$

```
Theorem sigma: forall n:nat, (f n) = (g n).
Proof.
  elim.                 (* n に対する数学的帰納法で証明 *)
  by [compute].         (* n=0 のときは計算で明らか *)
  move => n H.
  rewrite /f.
  fold f.
  rewrite H.            (* n のときの仮定を使うと *)
  rewrite /g.
  apply lemma4.         (* lemma4 の計算から証明できる *)
Qed.
```

Institute of Mathematics for Industry
Kyushu University

証明された関数を実際に使う (1)

性質が証明された実装関数 f と g を Extraction を用いて, ファイル "sigma.hs" に Haskell 言語の関数として出力する.
出力前のおまじないとして, いくつか (list, symbol, bool, nat) の型定義が必要.

```
Extraction Language Haskell.

Extract Inductive list => "([])" ["[]" "(:)"].
Extract Inductive sumbool => "Prelude.Bool"
     ["Prelude.True" "Prelude.False"].
Extract Inductive bool => "Prelude.Bool"
     ["Prelude.True" "Prelude.False"].
Extract Inductive nat => "Prelude.Int"
     ["0" "Prelude.succ"]
   "(\f0 fS n -> if (n Prelude.== 0)
         then f0 () else fS (n Prelude.- 1))".

Extraction "sigma.hs" f g.
```

Institute of Mathematics for Industry
Kyushu University

証明された関数を実際に使う (2)

出力されたファイル sigma.hs 中の 1 行

```
import qualified GHC.Base
```

を型定義文の上へ移動する必要がある.

```
import qualified Prelude

import qualified GHC.Base <-- (ここに移動)

unsafeCoerce :: a -> b
#ifdef __GLASGOW_HASKELL__
-- import qualified GHC.Base <-- (この行を移動)
unsafeCoerce = GHC.Base.unsafeCoerce#
#else
-- HUGS
import qualified IOExts
unsafeCoerce = IOExts.unsafeCoerce
#endif
```

Institute of Mathematics for Industry
Kyushu University

証明された関数を実際に使う (3)

出力された Haskell 言語の関数を Glasgow Haskell Interpreter[1] で実行してみる.

```
ym$ ghci sigma.hs
GHCi, version 7.6.3: http://www.haskell.org/ghc/  :? for help
[1 of 1] Compiling Sigma             ( sigma.hs, interpreted )
Ok, modules loaded: Sigma.
*Sigma> f 5
15
*Sigma> g 5
15
*Sigma> :q
```

※ 関数 f と関数 g の値が常に等しいことは証明されています!
具体的なデータに対する試験 (テスト) での結果の確認は不要です.

[1] https://www.haskell.org/ghc/

Institute of Mathematics for Industry
Kyushu University

証明された関数を実際に使う (4)

自動生成された証明済の Haskell 言語のプログラム.

```
f :: Int -> Int
f n =
  (\f0 fS n -> if (n == 0) then f0 () else fS (n - 1))
    (\_ ->
    0)
    (\p ->
    addn (succ p) (f p))
    n
g :: Int -> Int
g n =
  divn (muln n (succ n)) (succ (succ 0))
```

Institute of Mathematics for Industry
Kyushu University

自然数上の命題 (1)

与えられた自然数 x に対して,
真 (True) か偽 (False) を返す関数の例として E1 を考える.

```
Definition E1 (x:nat):Prop :=
match x with
| 0    => False
| n.+1 => match n with
          | 0    => True
          | _.+1 => False
          end
end.
Compute [:: (E1 0); (E1 1); (E1 2); (E1 3)].
 = [:: False; True; False; False]
```

真偽を返す関数は自然数上の命題でもあり,
命題 (E1 x) は, $x = 1$ のときだけ真で $x \neq 1$ のときは偽である.

Institute of Mathematics for Industry
Kyushu University

否定の証明 (1)

- Coq では否定は False を導く命題として定義される.
- $\neg P$ は $P \to$ False と考える.
- False は何でも証明出来る命題として定義されている.
- 何でも証明できるという証明の名前は False_rect である.
- (False_rect P) は False → P の証明である.

```
Lemma lemma5: (E1 0) -> False.
Proof.
  compute.
  apply (False_rect False).
Qed.
```

Institute of Mathematics for Industry
Kyushu University

否定の証明 (2)

- 命題 $1 \neq 0$ は, $(1 = 0) \to$ False である.
- (E1 1) は計算で証明出来る.
- 仮定の $1 = 0$ から, (E1 1) $\to$ (E1 0) が証明出来る.
- (E1 0) $\to$ False を lemma5 で証明した.
- 従って, $(1 = 0) \to$ False が証明出来る.

```
Lemma lemma6: (1 = 0) -> False.
Proof.
  have: (E1 1).
  compute.
  apply I.
  move => H1 H2; move: H1.
  rewrite H2.
  apply lemma5.
Qed.
```

Institute of Mathematics for Industry
Kyushu University

自然数上の命題 (2)

関数 NE1 を以下のようにする.

```
Definition NE1 (x:nat):Prop :=
match x with
| 0    => True
| n.+1 => match n with
          | 0    => False
          | _.+1 => True
          end
end.
```

このとき,

Lemma

$$\forall x, \neg(\mathrm{E1}\ x) \leftrightarrow (\mathrm{NE1}\ x)$$

は殆ど計算だけで証明出来る.

Institute of Mathematics for Industry
Kyushu University

自然数上の命題 (3)

自然数上の命題は真偽を計算する関数で与えるだけでなく
帰納的な定義も出来る.

```
Inductive even: nat -> Prop :=
|even_0:  even 0
|even_SS: forall n, (even n) -> (even (n.+2)).
```

言葉で書けば,

1. $\mathbf{0}$ は偶数である.
2. n が偶数ならば, $n + 2$ は偶数である.

で, 命題「x は偶数である」を定義出来る.
1. の証明の名前が even_0, そして, 2. の証明の名前が even_SS である.

Institute of Mathematics for Industry
Kyushu University

帰納的に定義された命題の証明

Lemma

2 は偶数である

を証明する.

```
(even_SS n): (even n) -> (even (n.+2))
```

なのと, **0. + 2 = 2** の計算が出来るので, `(even_SS 0)` は

```
(even 0) -> (even 2)
```

の証明である. `(even 0)` は `even_0` で証明されているので, `((even_SS 0) even_0)` が, `(even 2)` の証明になる.

```
Lemma lemma7: (even 2).
Proof.
  apply (even_SS 0 even_0).
Qed.
```

Institute of Mathematics for Industry
Kyushu University

帰納的に定義された命題の否定命題の証明 (1)

Lemma

1 は偶数でない

の証明を考える. ¬(even 1) は, (even 1) → False である. (even n) を帰納的に証明する証明 even_0, even_SS のように, ¬(even n) を帰納的に構成する odd_1, odd_SS のような証明を準備して計算で証明する方針も考えられる. 偶数の集合の場合は補集合である奇数の集合を帰納的に定義出来るが, 一般には任意の帰納的可算集合の補集合を帰納的可算集合として定義することは出来ない.

任意の帰納的可算集合が帰納的集合であるわけではない.

Institute of Mathematics for Industry
Kyushu University

帰納的に定義された命題の否定命題の証明 (2)

帰納的に定義された命題 even を関数で定義された命題 NE1 と関係付けることにより, 計算による性質の証明が可能になる.

```
Lemma lemma8: forall n, (even n) -> (NE1 n).
Proof.
  move => n.
  case.          (* even の構成の帰納法で証明 *)
  compute.       (* (even 0) のときは計算で明らか *)
  apply I.
  move => n0 H.
  compute.       (* (even (n.+2)) のときも計算で明らか *)
  apply I.
Qed.
```

(even n) を仮定した命題は, even の構成の帰納法によって証明する. このとき, (NE1 n.+2)=True は計算で証明出来ることに注意する.

Institute of Mathematics for Industry
Kyushu University

帰納的に定義された命題の否定命題の証明 (3)

(NE1 1) → False は計算で証明出来るので先の補題 lemma8 で $n = 1$ と置いた場合を考えて, (even 1) → False が証明出来る.

```
Lemma lemma9: (even 1) -> False.
Proof.
  apply (lemma8 1).
Qed.
```

実は, inversion という Tactic は, NE1 のような関数を自動生成してくれる.

```
Lemma lemma10: (even 1) -> False.
Proof.
  move => H.
  inversion H.
Qed.
```

Institute of Mathematics for Industry
Kyushu University

まとめ

- 本来, 定理証明器は正しいプログラムを書くために開発されて来た.
- 数式や論理式で記述されたプログラムの正しさは数式や論理式の正しさ, すなわち, 数学の命題の正しさに含まれる.
- 近年は検証の難しい数学の定理の証明検証にも定理証明器が利用されている.
- 形式証明で利用しやすい命題とそうでない命題がある.
- 同値, あるいは, 包含関係のある利用しやすい命題が必要.
- 要. 数学理論の再編成. (Abstract が Nonsense でなくなった!)
- 証明検証しやすい作法でプログラムを書く事も大切.
- 基本的にプログラムは実応用のために書かれるものであり, 一般的には, それそのものを楽しんだり, 鑑賞したりするものではない.
- 数学の証明は鑑賞したり, それそのものを楽しむことも考えられる.
- 数学とプログラムの中間に定理証明器がいる.

Institute of Mathematics for Industry
Kyushu University

参考文献 (1)

- Reynald Affeldt, 定理証明支援系 Coq 入門, 日本ソフトウェア科学会チュートリアル, 2014.
  https://staff.aist.go.jp/reynald.affeldt/ssrcoq/
- Adam Chlipala, Certified Programming with Dependent Types, MIT Press, 2013.
  http://adam.chlipala.net/cpdt/
- G.Gonthier et.al., A Small Scale Reflection Extension for the Coq system, Inria Research Report, 2014.
  https://hal.inria.fr/inria-00258384/en
- G.Gonthier, R. Stéphane Le, An Ssreflect Tutorial, 2009.
  https://hal.inria.fr/inria-00407778/file/RT-367.pdf
- The Coq Development Team, The Coq Proof Assistant Reference Manual, Ver.8.4pl5, 2014.
  https://coq.inria.fr/distrib/V8.4pl5/files/Reference-Manual.pdf

Institute of Mathematics for Industry
Kyushu University

参考文献 (2)

- TPP2014, 高信頼な理論と実装のための定理証明および定理証明器, 九州大学, 2014. http://imi.kyushu-u.ac.jp/lasm/tpp2014/index_ja.html
- Y. Bertot, Interactive Theorem Proving and Program Development, Springer, 2004. (Coq'Art)
- The Univelent Foundations Program, Homotopy Type Theory: Univalent Foundations of Mathematics, http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.
- 溝口佳寛, 田上真, ケプラー予想の計算機による証明と検証について, 数学セミナー 12 月号, 48–54, 2014. http://bit.ly/KeplerConjecture
- 溝口佳寛他, 有限オートマトンとスティッカー系に関する Coq による形式証明について, 日本数学会年会講演スライド, 2014. http://www.slideshare.net/yoshihiromizoguchi/coq-32356516

Institute of Mathematics for Industry
Kyushu University