

Coq 練習帳 (第 0.1 版)

溝口 佳寛

(九州大学マス・フォア・インダストリ研究所)

2015/06/20

Email: ym@imi.kyushu-u.ac.jp

目次

第 1 章	Library 01CertifiedFunction	4
1.1	第 1 回「プログラム関数の同値性の証明」	4
1.2	自然数の性質	4
1.3	関数定義	7
1.4	Haskell 関数出力	9
第 2 章	Library 02DeMorgansLaw	11
2.1	第 2 回「ドモルガンの法則の証明」	11
2.2	有限の場合	11
2.2.1	直和 (coproduct)	11
2.2.2	$(\neg A_1) \vee (\neg A_2) \rightarrow \neg(A_1 \wedge A_2)$	13
2.2.3	直積 (product)	14
2.2.4	$\neg(A_1 \vee A_2) \rightarrow (\neg A_1) \wedge (\neg A_2)$	15
2.2.5	$(\neg A_1) \wedge (\neg A_2) \rightarrow \neg(A_1 \vee A_2)$	16
2.2.6	$\neg(A_1 \wedge A_2) \rightarrow (\neg A_1) \vee (\neg A_2)$	17
2.3	無限の場合	18
2.3.1	$\exists x, \neg P(x) \rightarrow \neg(\forall x, P(x))$	18
2.3.2	$\neg(\exists x, P(x)) \rightarrow \forall x, \neg P(x)$	20
2.3.3	$\forall x, \neg P(x) \rightarrow \neg(\exists x, P(x))$	21
第 3 章	Library 03ElementaryTactics	22
3.1	第 3 回「証明の基本」	22
3.2	等式の証明 (rewrite)	22
3.2.1	erefl	22
3.2.2	eq_ind	23
3.2.3	rewrite	23
3.3	構成と反対方向の性質 (inversion)	24
3.3.1	nat	24
3.3.2	even	26
3.4	否定の証明 (discriminate, inversion)	27
3.4.1	False	27
3.4.2	$x \neq y$ (discriminate)	28
3.5	$\neg P(a)$ (inversion)	29
第 4 章	Library 04TPPmark2014	32
4.1	第 4 回「TPPmark2014 を解く」	32
4.2	補題	32
4.3	帰納法の復習	37
4.3.1	通常の帰納法	37

4.3.2	超限歸納法 (1)	38
4.3.3	超限歸納法 (2)	40
4.4	命題	40

はじめに

Coq/Ssreflect のインストール

2015 年 6 月 20 日現在の情報です.

Coq8.5beta2, ssreflect-1.5.coq85beta2, mathcomp-1.5.coq85beta2 が公表されています.

- <https://coq.inria.fr/coq-85>
- <http://ssr.msr-inria.inria.fr/FTP/>

それぞれの OS ごとの情報は以下の通りです.

- Mathlibre (Linux)
ライブ DVD に, CoqIDE, coq8.4pl3, ssreflect1.5, mathcomp1.5 が入っています. すぐに使えます.
<http://mirror.math.kyushu-u.ac.jp/mathlibre/mathlibre-debian-amd64-20150303-ja.iso>
- Windows
Coq8.5beta2, ssreflect1.5 と mathcomp1.5 のインストーラがあります.
 - <https://coq.inria.fr/distrib/V8.5beta2/files/coq-installer-8.5beta2.exe>
 - <http://ssr.msr-inria.inria.fr/FTP/ssr-mathcomp-installer-1.5coq8.5beta2.exe>
- MacOSX
CoqIDE8.5beta1 のパッケージには ssreflect1.5, mathcomp1.5 が入っています.
https://coq.inria.fr/distrib/V8.5beta1/files/coqide-8.5beta1_MathComp-1.5.dmg
(注) CoqIDE8.5beta2 のパッケージには ssreflect/MathComp は入っていません. 自分でコンパイルして実装する必要があります.

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.

- ProofGeneral <http://proofgeneral.inf.ed.ac.uk/>
- Coq/jEdit <http://coqide.bitbucket.org/>

第1章 Library 01CertifiedFunction

1.1 第1回「プログラム関数の同値性の証明」

今回は、 $1 + 2 + \dots + n$ 繰り返し計算 (再帰呼び出し) で計算する関数 $sumA(n) = \sum_{k=0}^n k$ と総和の公式を用いて計算する関数 $sumB(n) = \frac{n(n+1)}{2}$ に対して、2つの関数の値が等しいこと、すなわち

Theorem 1.1 *Let $n \in \mathbb{N}$. Define two functions $sumA(n)$ and $sumB(n)$ as follows:*

$$\begin{aligned} sumA(n) &= \sum_{k=0}^n k, \\ sumB(n) &= \frac{n(n+1)}{2}. \end{aligned}$$

Then, $sumA(n) = sumB(n)$ for any n .

Where $\sum_{k=0}^0 k = 0$ and $\sum_{k=0}^{n+1} k = (n+1) + \sum_{k=0}^n k$ for $n \in \mathbb{N}$.

を証明する.

そして、証明された関数 $sumA(n)$ と $sumB(n)$ を Haskell 言語の関数として出力する. この2つの Haskell 関数は同じ値を返すことが形式的に証明されていることになる.

Require Import *Ssreflect.ssreflect Ssreflect.ssrbool Ssreflect.ssrfun Ssreflect.ssrnat MathComp.div List.*

1.2 自然数の性質

自然数の集合は帰納的に定義される.

```
Inductive nat : Set :=
| 0 : nat
| S : nat -> nat.
```

加法や乗法などの自然数上の演算は帰納的に定義される.

```
Fixpoint plus (n m:nat) : nat :=
  match n with
  | 0 => m
  | S p => S (p + m)
  end
```

わかりやすい記法を定義する.

```

Notation succn := Datatypes.S.
Notation "n .+1" := (succn n)
Notation "m + n" := (plus m n)

```

加法や乗法の性質は帰納的に証明される. 以下に, 基本的な補題の例を列挙する.

```

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.

```

`add0n`, `addnS` と `addSn` について, 自分で証明してみる. 関数定義に従い式を展開することで証明出来る.

Lemma *my_add0n*: $\forall n:nat, 0 + n = n$.

Proof.

```

move => n.
rewrite /addn/addn_rec/plus.
apply erefl.

```

Restart.

by [].

Qed.

Lemma *my_addSn*: $\forall m n:nat, m.+1 + n = (m + n).+1$.

Proof.

```

move => m n.
rewrite /addn/addn_rec/plus.
fold plus.
apply erefl.

```

Restart.

by [].

Qed.

Lemma *my_addnS*: $\forall m n:nat, m + n.+1 = (m + n).+1$.

Proof.

```

elim.
move => n.
rewrite !add0n.
apply erefl.
move => n H n0.
rewrite !addSn.
rewrite H.
apply erefl.

```

Restart.

by [].

Qed.

次に, 交換法則の証明である `addnC` を `my_addnC` という名前で, 自分で帰納法を使って証明してみる.

Lemma *my_addnC*: $\forall n\ m:\text{nat}, m + n = n + m$.

Proof.

```
move  $\Rightarrow$  n m.
elim m.
rewrite addn0 add0n.
apply erefl.
move  $\Rightarrow$  n0 H1.
rewrite addSn.
rewrite H1.
rewrite -addnS.
apply erefl.
```

Restart.

```
move  $\Rightarrow$  n m.
ring.
```

Qed.

Lemma で定める名前は命題の名前ではなく、証明の名前です。命題の名前は **Definition** で定めます。

Definition *my_addC* := $\forall n\ m:\text{nat}, m + n = n + m$.

Print *my_addC*.

```
my_addC = forall n m : nat, m + n = n + m : Prop
```

Goal *my_addC*.

```
move  $\Rightarrow$  n m.
apply (addnC m n).
```

Qed.

Goal *my_addC*.

```
apply /my_addnC.
```

Qed.

一旦、自然数上で成立する演算の性質がわかると、その性質の定理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る。また、数式を簡潔な方向に変換して適用する定理も探して証明するタクティクと呼ばれる手続きがある (**ring** など)。

Lemma *lemma1*: $\forall n:\text{nat}, (n+1)*2 + (n \times (n+1)) = (n+1 \times (n+2))$.

Proof.

```
move  $\Rightarrow$  n.
ring.
```

Qed.

```
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.
```

Lemma lemma2: $\forall n:\text{nat}, (n.+1 \times 2) \% 2 = n.+1$.

Proof.

```
move  $\Rightarrow$   $n$ .
rewrite -(muln_divA (n.+1)).
rewrite (divnn 2).
simpl.
by [rewrite muln1].
apply (dvdnn 2).
```

Qed.

Lemma lemma3: $\forall n:\text{nat}, 2 \% | (n.+1 \times 2)$.

Proof.

```
move  $\Rightarrow$   $n$ .
apply dvdn_mull.
apply (dvdnn 2).
```

上記2行は, `apply (@dvdn_mull 2 (n.+1) 2 (dvdnn 2))`. でも良い.

Qed.

Lemma lemma4: $\forall n:\text{nat}, n.+1 + (n \times n.+1) \% 2 = (n.+1 \times n.+2) \% 2$.

Proof.

```
move  $\Rightarrow$   $n$ .
rewrite -lemma1.
rewrite (divnDl (n×n.+1) (lemma3 n)).
by [rewrite lemma2].
```

Qed.

1.3 関数定義

ここでは, 関数 $\text{sumA}(n)$ と $\text{sumB}(n)$ の定義を行う. 定義の後, Coq システム内でも `Compute` で関数の値の例を計算して確認しておく.

```
Fixpoint sumA (n:nat) :=
match n with
| 0  $\Rightarrow$  0
| p.+1  $\Rightarrow$  (p.+1) + sumA p
end.
```

Compute (sumA 3).

Compute (sumA 5).

```
Compute (sumA 3).
= 6
: nat
Compute (sumA 5).
= 15
: nat
```


Definition *sumB* (*n:nat*) := ((*n* × (*n*.+1)) %/ 2).

Compute (*sumB* 3).

Compute (*sumB* 5).

```
Compute (sumB 3).  
  = 6  
  : nat  
Compute (sumB 5).  
  = 15  
  : nat
```

Set Printing All とすることで、記号 (*や%/など) が定義関数名で表示される。例えば, %/は `divn` という関数で定義されていることがわかる。また, **Print half** などとすることで関数定義を見ることが出来る。

Print *sumA*.

Print *sumB*.

Set Printing All.

Print *sumB*.

Unset Printing All.

Print *half*.

```
sumA =  
fix sumA (n : nat) : nat := match n with  
  | 0 => 0  
  | p.+1 => p.+1 + sumA p  
end  
  
  : nat -> nat  
sumB = fun n : nat => (n * n.+1) %/ 2  
  : nat -> nat  
sumB = fun n : nat => divn (muln n (S n)) (S (S 0))  
  : nat -> nat  
half =  
fix half (n : nat) : nat :=  
  match n with  
  | 0 => n  
  | n'.+1 => uphalf n'  
end  
with uphalf (n : nat) : nat :=  
  match n with  
  | 0 => n  
  | n'.+1 => (half n').+1  
end  
for half  
  : nat -> nat
```

上で定義した補題たちを使って, 最後に主定理を証明する.

Theorem *sumAB*: $\forall n:\text{nat}, (\text{sumA } n) = (\text{sumB } n)$.

Proof.

move $\Rightarrow n$.

elim *n*.

by [compute].

move $\Rightarrow n0\ H$.

rewrite /*sumA*.

fold *sumA*.

elim *n*; by [compute]; move $\Rightarrow n0\ H$. は, elim *n* $\Rightarrow [//n0\ H].$ と書ける. rewrite /*sumA*; fold *sumA* は, simpl で行える. rewrite *H*.

rewrite /*sumB*.

apply *lemma4*.

Qed.

1.4 Haskell 関数出力

証明された関数 *sumA* と *sumB* を **Extraction** を用いて, ファイル "*sumA.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. そして, 出力されたファイル中の 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 <-- (Moved to)
unsafeCoerce :: a -> b
#ifdef __GLASGOW_HASKELL__
-- import qualified GHC.Base <-- (Moved from)
unsafeCoerce = GHC.Base.unsafeCoerce#
#else
-- HUGS
import qualified IOExts
unsafeCoerce = IOExts.unsafeCoerce
#endif
```

Extraction *Language Haskell*.

Extract Inductive *list* \Rightarrow "[()]" ["[]" "(:)"].

Extract Inductive *sumbool* \Rightarrow "Prelude.Bool" ["Prelude.True" "Prelude.False"].

Extract Inductive *bool* \Rightarrow "Prelude.Bool" ["Prelude.True" "Prelude.False"].

Extract Inductive *nat* \Rightarrow "Prelude.Int" ["0" "Prelude.succ"] "(\\fO fS n -> if (n Prelude.== 0) then fO () else fS (n Prelude.- 1))".

Extraction "*sumA.hs*" *sumA sumB*.

出力された Haskell 関数のファイルを実行してみる.

```

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

```

出力された Haskell 関数定義文は, 以下のようになる.

```

sumA :: Prelude.Int -> Prelude.Int
sumA n =
  (\f0 fS n -> if (n Prelude.== 0) then f0 () else fS (n Prelude.- 1))
    (\_ ->
      0)
    (\p ->
      addn (Prelude.succ p) (sumA p))
    n
sumB :: Prelude.Int -> Prelude.Int
sumB n =
  divn (muln n (Prelude.succ n)) (Prelude.succ (Prelude.succ 0))

```

第2章 Library 02DeMorgansLaw

2.1 第2回「ドモルガンの法則の証明」

今回は、ブール演算の基本的な性質である「ドモルガンの法則」の証明を行う。有限の場合 (2 つの集合) のドモルガンの法則は以下の 4 つになる。

$$(1) (\neg A_1) \vee (\neg A_2) \rightarrow \neg(A_1 \wedge A_2)$$

$$(2) \neg(A_1 \vee A_2) \rightarrow (\neg A_1) \wedge (\neg A_2)$$

$$(3) (\neg A_1) \wedge (\neg A_2) \rightarrow \neg(A_1 \vee A_2)$$

$$(4) \neg(A_1 \wedge A_2) \rightarrow (\neg A_1) \vee (\neg A_2)$$

(1) の逆の (4) の証明のみ排中律が必要となる。2.2 節において, Lemma DeMorgan01 で (1) を Lemma Demorgan02 で (2) をその逆の (3) を Lemma Demorgan03 で示す。そして, Lemma Demorgan04 で (4) を示す。2.3 節においては, 無限の族に対するドモルガンの法則の中で排中律の不要な以下の 3 つの命題を証明する。

$$(1) \exists x, \neg P(x) \rightarrow \neg(\forall x, P(x))$$

$$(2) \neg(\exists x, P(x)) \rightarrow \forall x, \neg P(x)$$

$$(3) \forall x, \neg P(x) \rightarrow \neg(\exists x, P(x))$$

Lemma DeMorgan05 で (1) を Lemma DeMorgan06 で (2) を Lemma DeMorgan07 で (3) を示す。

Require Import Ssreflect.ssreflect Ssreflect.ssrbool Ssreflect.ssrfun.

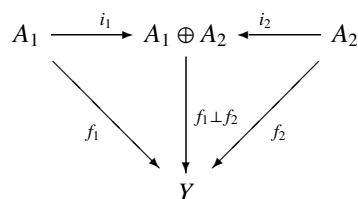
2.2 有限の場合

Section DeMorgans_Laws.

Variables (A:Set) (P Q:A → Prop).

Variables (A1 A2:Prop).

2.2.1 直和 (coproduct)



集合 A_1, A_2 の直和集合は

$$A_1 \oplus A_2 = \{(x_1, 1) \mid x_1 \in A_1\} \cup \{(x_2, 2) \mid x_2 \in A_2\}$$

で定義される. ここで, $i_k : A_k \rightarrow A_1 \oplus A_2$ を $i_k(x_k) = (x_k, k)$ とする. 集合 Y に対して, 次の同型関係がある.

$$[A_1 \rightarrow Y] \times [A_2 \rightarrow Y] \cong [A_1 \oplus A_2 \rightarrow Y]$$

関数 $f_1 : A_1 \rightarrow Y, f_2 : A_2 \rightarrow Y$ に対して, $f_1 \perp f_2((x_k, k)) = f_k(x_k)$ とし, この 1 対 1 対応を与える同型射の左から右への関数を `or_intro`, 右から左への関数を `or_case` とする.

$$\text{or_intro} : [A_1 \rightarrow Y] \times [A_2 \rightarrow Y] \rightarrow [A_1 \oplus A_2 \rightarrow Y]$$

$$\text{or_case} : [A_1 \oplus A_2 \rightarrow Y] \rightarrow [A_1 \rightarrow Y] \times [A_2 \rightarrow Y]$$

ここで, $\text{or_intro}(f_1, f_2) = f_1 \perp f_2$ であり, $\text{or_case}(g) = (g \circ i_1, g \circ i_2)$ である.

直和 $X_1 \vee X_2$ は Colimit なので, 関数の対 $(f_1 : X_1 \rightarrow Y, f_2 : X_2 \rightarrow Y)$ と関数 $f_1 \perp f_2 : X_1 \vee X_2 \rightarrow Y$ が, 1 対 1 に対応する. $f_1 \perp f_2$ を与える関数を `or_intro` とする.

Lemma *or_intro*: $\forall X1 X2 Y:\text{Prop}, ((X1 \rightarrow Y) \times (X2 \rightarrow Y)) \rightarrow (X1 \vee X2) \rightarrow Y.$

Proof.

`move \Rightarrow $X1 X2 Y H$.`

`elim $:H$.`

`move \Rightarrow $H1 H2 H3$.`

`case $:H3$.`

`apply $H1$.`

`apply $H2$.`

Qed.

Lemma *or_case*: $\forall X1 X2 Y:\text{Prop}, ((X1 \vee X2) \rightarrow Y) \rightarrow ((X1 \rightarrow Y) \times (X2 \rightarrow Y)).$

Proof.

`move \Rightarrow $X1 X2 Y H$.`

`have $:X1 \rightarrow X1 \vee X2$.`

`move \Rightarrow $X11$.`

`left.`

`apply $X11$.`

`move \Rightarrow $X11$.`

`have $:X2 \rightarrow X1 \vee X2$.`

`move \Rightarrow $X22$.`

`right.`

`apply $X22$.`

`move \Rightarrow $X22$.`

`apply pair.`

`move \Rightarrow $X111$.`

`apply $(H (X11 X111))$.`

`move \Rightarrow $X222$.`

`apply $(H (X22 X222))$.`

Qed.

2.2.2 $(\neg A_1) \vee (\neg A_2) \rightarrow \neg(A_1 \wedge A_2)$

次のドモルガンの法則を証明する.

Theorem 2.1 $(\neg A_1) \vee (\neg A_2) \rightarrow \neg(A_1 \wedge A_2)$

直和からの関数なので, 構成する 2 つの関数たち $(\neg A_1) \rightarrow \neg(A_1 \wedge A_2)$ と $(\neg A_2) \rightarrow \neg(A_1 \wedge A_2)$ を先に考える. $\neg A_1$ は関数 $f_1 : A_1 \rightarrow \text{False}$, $\neg(A_1 \wedge A_2)$ は関数 $f_p : (A_1 \wedge A_2) \rightarrow \text{False}$ であるので, f_1 が与えられたときに, f_p を作る方法を考えれば良い. 直積 $A_1 \wedge A_2$ には, 射影 $\text{proj}_1 : (A_1 \wedge A_2) \rightarrow A_1$ が存在するので, これと f_1 合成することで f_p を得ることが出来る. すなわち, $f_p := f_1 \circ \text{proj}_1$ とすれば良い.

Lemma DeMorgan011: $\neg A_1 \rightarrow \neg(A_1 \wedge A_2)$.

Proof.

move $\Rightarrow f1$.

move $\Rightarrow xy$.

apply $(f1 (\text{proj1 } xy))$.

Restart.

move $\Rightarrow f1 \ xy$.

elim xy .

move $\Rightarrow a1 \ a2$.

apply $f1$.

apply $a1$.

Qed.

Lemma DeMorgan012: $\neg A_2 \rightarrow \neg(A_1 \wedge A_2)$.

Proof.

move $\Rightarrow f2 \ xy$.

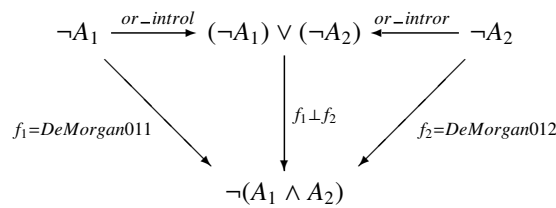
apply $(f2 (\text{proj2 } xy))$.

Qed.

出来上がった関数 $\text{DeMorgan011} : \neg A_1 \rightarrow \neg(A_1 \wedge A_2)$, $\text{DeMorgan012} : \neg A_2 \rightarrow \neg(A_1 \wedge A_2)$ に対して,

$$\text{DeMorgan011} \perp \text{DeMorgan012} : (\neg A_1) \vee (\neg A_2) \rightarrow \neg(A_1 \wedge A_2)$$

がドモルガンの法則の証明に対応する関数である. この関数は, or_intro を用いて構成される.



Lemma DeMorgan01 : $\neg A_1 \vee \neg A_2 \rightarrow \neg(A_1 \wedge A_2)$.

Proof.

apply $(\text{or_intro } (\neg A_1) (\neg A_2) (\neg(A_1 \wedge A_2)) (\text{pair DeMorgan011 DeMorgan012}))$.

Qed.

Print or_intro.

Print DeMorgan011.

Print DeMorgan012.

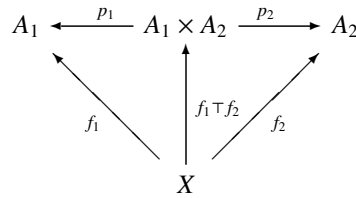
Print *DeMorgan01*.

```

DeMorgan011 =
fun (f1 : ~ A1) (xy : A1 /\ A2) => f1 (proj1 xy)
  : ~ A1 -> ~ (A1 /\ A2)
DeMorgan012 =
fun (f2 : ~ A2) (xy : A1 /\ A2) => f2 (proj2 xy)
  : ~ A2 -> ~ (A1 /\ A2)
DeMorgan01 =
or_intro (~ A1) (~ A2) (~ (A1 /\ A2)) (DeMorgan011, DeMorgan012)
  : ~ A1 \/ ~ A2 -> ~ (A1 /\ A2)

```

2.2.3 直積 (product)



集合 A_1, A_2 の直積は

$$A_1 \times A_2 = \{(x_1, x_2) \mid x_1 \in A_1, x_2 \in A_2\}$$

で定義される. ここで, $p_k : A_1 \times A_2 \rightarrow A_k$ を $p_k(x_1, x_2) = x_k$ とする.

集合 X に対して, 次の同型関係がある.

$$[X \rightarrow A_1] \times [X \rightarrow A_2] \cong [X \rightarrow A_1 \times A_2]$$

関数 $f_1 : X \rightarrow A_1, f_2 : X \rightarrow A_2$ 対して, $f_1 \top f_2(x) = (f_1(x), f_2(x))$ とし, この 1 対 1 対応を与える同型射の左から右への関数を **prod_intro**, 右から左への関数を **prod_case** とする.

$$\text{prod_intro} : [X \rightarrow A_1] \times [X \rightarrow A_2] \rightarrow [X \rightarrow A_1 \times A_2]$$

$$\text{prod_case} : [X \rightarrow A_1 \times A_2] \rightarrow [X \rightarrow A_1] \times [X \rightarrow A_2]$$

ここで, $\text{prod_intro}(f_1, f_2) = f_1 \top f_2$ であり, $\text{prod_case}(g) = (p_1 \circ g, p_2 \circ g)$ である.

直積 $X_1 \wedge X_2$ は Limit なので, 関数の対 $(f_1 : X \rightarrow X_1, f_2 : X \rightarrow X_2)$ と関数 $f_1 \top f_2 : X \rightarrow X_1 \wedge X_2$ が, 1 対 1 に対応する. $f_1 \top f_2$ を与える関数を **and_intro** とする.

Lemma and_intro: $\forall X1 X2 X:\text{Prop},$
 $((X \rightarrow X1) \times (X \rightarrow X2)) \rightarrow (X \rightarrow (X1 \wedge X2)).$

Proof.

move $\Rightarrow X1 X2 X H.$

elim $:H.$

move $\Rightarrow X11 X22 XX.$

split.

apply $(X11 XX).$

apply (X22 XX).

Qed.

Lemma and_case: $\forall X1 X2 X:\text{Prop},$

$(X \rightarrow (X1 \wedge X2)) \rightarrow ((X \rightarrow X1) \times (X \rightarrow X2)).$

Proof.

move $\Rightarrow X1 X2 X H.$

split.

move $\Rightarrow XX.$

apply (proj1 (H XX)).

move $\Rightarrow XX.$

apply (proj2 (H XX)).

Qed.

2.2.4 $\neg(A_1 \vee A_2) \rightarrow (\neg A_1) \wedge (\neg A_2)$

もうひとつのドモルガンの法則は以下のように定式化される.

Theorem 2.2 $\neg(A_1 \vee A_2) \rightarrow (\neg A_1) \wedge (\neg A_2)$

Lemma DeMorgan021: $\neg(A1 \vee A2) \rightarrow \neg A1.$

Proof.

move $\Rightarrow H H1.$

apply (H (or_introl H1)).

Qed.

Lemma DeMorgan022: $\neg(A1 \vee A2) \rightarrow \neg A2.$

Proof.

move $\Rightarrow H H1.$

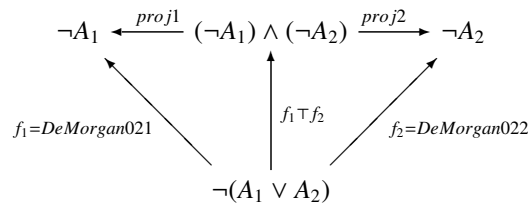
apply (H (or_intror H1)).

Qed.

出来上がった関数 $DeMorgan021 : \neg(A_1 \vee A_2) \rightarrow \neg A_1$, $DeMorgan022 : \neg(A_1 \vee A_2) \rightarrow \neg A_2$ に対して,

$$DeMorgan021 \top DeMorgan022 : \neg(A_1 \vee A_2) \rightarrow (\neg A_1 \wedge \neg A_2)$$

がドモルガンの法則の証明に対応する関数である. この関数は `and_intro` を用いて証明される.



Theorem DeMorgan02: $\neg(A1 \vee A2) \rightarrow \neg A1 \wedge \neg A2.$

Proof.

apply (and_intro (¬ A1) (¬ A2) (¬ (A1 ∨ A2)) (pair DeMorgan021 DeMorgan022)).

Qed.

Print *DeMorgan021*.

Print *DeMorgan022*.

Print *DeMorgan02*.

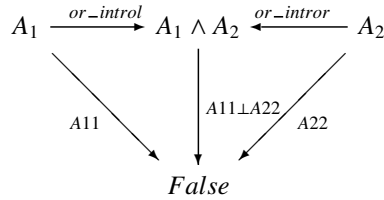
```
DeMorgan021 =
fun (H : ~ (A1 ∨ A2)) (H1 : A1) => H (or_introl H1)
  : ~ (A1 ∨ A2) -> ~ A1
DeMorgan022 =
fun (H : ~ (A1 ∨ A2)) (H1 : A2) => H (or_intror H1)
  : ~ (A1 ∨ A2) -> ~ A2
DeMorgan02 =
and_intro (~ A1) (~ A2) (~ (A1 ∨ A2)) (DeMorgan021, DeMorgan022)
  : ~ (A1 ∨ A2) -> ~ A1 ∧ ~ A2
```

2.2.5 $(\neg A_1) \wedge (\neg A_2) \rightarrow \neg(A_1 \vee A_2)$

次に, Theorem 2.2 の逆の証明を考える.

Theorem 2.3 $(\neg A_1) \wedge (\neg A_2) \rightarrow \neg(A_1 \vee A_2)$

$\neg A_1, \neg A_2$ の証明は, それぞれ, *False* への関数, $f_1 : A_1 \rightarrow \text{False}$, $f_2 : A_2 \rightarrow \text{False}$ である. 求めたい証明は, $A_1 \vee A_2$ から *False* への関数なので, これは, *or_intro* を用いて構成出来る. この方針で次の定理 *DeMorgan03* を証明する.



Theorem *DeMorgan03*: $\neg A1 \wedge \neg A2 \rightarrow \neg (A1 \vee A2)$.

Proof.

move $\Rightarrow H1$.

case *H1*.

move $\Rightarrow A11\ A22\ H2$.

apply ((*or_intro* *A1* *A2* *False* (*pair* *A11* *A22*))) *H2*).

Qed.

Print *DeMorgan03*.

```
DeMorgan03 =
fun H1 : ~ A1 ∧ ~ A2 =>
match H1 with
| conj A11 A22 => [eta or_intro A1 A2 False (A11, A22)]
end
  : ~ A1 ∧ ~ A2 -> ~ (A1 ∨ A2)
```

2.2.6 $\neg(A_1 \wedge A_2) \rightarrow (\neg A_1) \vee (\neg A_2)$

排中律の逆向きの証明は二重否定の仮定が必要である. 二重否定を仮定すると, ドモルガンの法則 (Theorem 2.3) の逆を証明出来る.

Theorem 2.4 *If $\forall A, \neg\neg A \rightarrow A$, then we have $\neg(A_1 \wedge A_2) \rightarrow (\neg A_1) \vee (\neg A_2)$.*

まず, 二重否定を仮定した排中律の証明を行う.

Lemma excluded : $\forall A0, (\forall A1, \neg\neg A1 \rightarrow A1) \rightarrow (A0 \vee \neg A0)$.

Proof.

move $\Rightarrow A0\ H2$.

apply $H2$.

move $\Rightarrow H3$.

apply $H3$.

right.

move $\Rightarrow H4$.

apply $H3$.

left.

apply $H4$.

Qed.

Lemma ContraPositive: $\forall P0\ Q0 : \text{Prop}, (P0 \rightarrow Q0) \rightarrow (\neg Q0 \rightarrow \neg P0)$.

Proof.

move $\Rightarrow P0\ Q0\ PQ\ NQ\ PP0$.

apply $(NQ\ (PQ\ PP0))$.

Qed.

Theorem DeMorgan04 : $(\forall A1, \neg\neg A1 \rightarrow A1) \rightarrow \neg(A1 \wedge A2) \rightarrow (\neg A1 \vee \neg A2)$.

Proof.

move $\Rightarrow H1\ H2$.

move: $(\text{excluded}\ A1\ H1)$.

H2 : $\sim (A1 \wedge A2)$

=====

$A1 \wedge \sim A1 \rightarrow \sim A1 \wedge \sim A2$

上記の排中律を仮定したゴールに対して, **case** で場合分けを行う. **A1** を仮定した場合は, $\sim A2$, すなわち, 右側 (right) を示す. $\sim A1$ を仮定した場合は, 左側 (left) が明らかに成立する. **case** \Rightarrow **AA1** は仮定した **A1** または $\sim A1$ に名前を着ける. その後, **[right|by left]** と書けば, 後半, $\sim A1$ を仮定した証明は終わり, 前半, **A1** を仮定した場合のみ続けられれば良い.

case $\Rightarrow AA1$; **[right|by left]**.

move $\Rightarrow AA2$.

apply $(H2\ (\text{conj}\ AA1\ AA2))$.

case 以下の後半の証明は, 先に証明した対偶命題 `ContraPositive` を使えば,

```
apply (ContraPositive A2 (A1 /\ A2) (@conj A1 A2 AA1) H2).
```

で証明出来る. `conj:A1 -> A2 -> A1 /\ A2` に注意する.

別の方法では, case の行は,

```
case => AA1; [by right => AA2; apply H2|by left].
```

とすれば, 1 行で証明が完成する.

Qed.

Print DeMorgan04.

```
DeMorgan04 =
fun (H1 : forall A1 : Prop, ~ ~ A1 -> A1) (H2 : ~ (A1 /\ A2)) =>
ssr_have (A1 \/ ~ A1) (excluded A1 H1)
(fun H3 : A1 \/ ~ A1 =>
match H3 with
| or_introl AA1 => or_intror (fun AA2 : A2 => H2 (conj AA1 AA2))
| or_intror H4 => or_introl H4
end)
: (forall A1 : Prop, ~ ~ A1 -> A1) -> ~ (A1 /\ A2) -> ~ A1 \/ ~ A2
```

2.3 無限の場合

有限の場合のドモルガンの法則 の無限の場合への拡張を考える.

2.3.1 $\exists x, \neg P(x) \rightarrow \neg(\forall x, P(x))$

Theorem 2.5 $\exists x, \neg P(x) \rightarrow \neg(\forall x, P(x))$

$\exists x : A$ は無限直和に対応する. すなわち, Colimit である. 関数の族

$$\{f_a : \neg P(a) \rightarrow Y \mid a \in A\}$$

と関数 $\perp_{a \in A} f_a : (\exists x, \neg P(x)) \rightarrow Y$ が 1 対 1 に対応する. $\perp_{a \in A} f_a$ を与える関数を `ex_intros` とする.

Lemma ex_intros: $\forall Y:\text{Prop},$

$(\forall x, ((P\ x) \rightarrow \text{False}) \rightarrow Y) \rightarrow ((\exists x, ((P\ x) \rightarrow \text{False})) \rightarrow Y).$

Proof.

move $\Rightarrow Y\ fx\ H1.$

case $H1.$

move $\Rightarrow x\ Px.$

apply $(fx\ x).$

apply $Px.$

Qed.

一方, $\forall x : A, P(x)$ は無限直積, すなわち Limit 対象である. その射影関数, 任意の $a \in A$ に対して, $P(a)$ を導く関数を `projx` とする.

Lemma *projx*: $\forall a:A, (\forall x:A, P\ x) \rightarrow (P\ a)$.

Proof.

`move \Rightarrow a fa.`

`apply (fa a).`

Qed.

ドモルガンの法則により直和からの関数を構成するための関数を定義する. すなわち, $a \in A$ に対して, 関数 $f_a : \neg P(a) \rightarrow \neg(\forall x, P(x))$ を作る. この関数を `DeMorgan050` として定義する.

Lemma *DeMorgan051*: $\forall a:A, \neg (P\ a) \rightarrow \neg (\forall x, (P\ x))$.

Proof.

`move \Rightarrow a.`

`move \Rightarrow fl.`

`move \Rightarrow fa.`

`apply (fl (projx a fa)).`

Qed.

Print *ex_intros*.

```
ex_intros =
fun (X Y : Prop) (fx : forall x : A, (P x -> X) -> Y)
  (H1 : exists x : A, P x -> X) =>
match H1 with
| ex_intro x Px => fx x Px
end
: forall X Y : Prop,
  (forall x : A, (P x -> X) -> Y) -> (exists x : A, P x -> X) -> Y
```

Check *DeMorgan051*.

Check *ex_intros*.

Check (*ex_intros* _ *DeMorgan051*).

```
DeMorgan050
: forall a : A, ~ P a -> ~ (forall x : A, P x)
ex_intros
: forall Y : Prop,
  (forall x : A, (P x -> False) -> Y) ->
  (exists x : A, P x -> False) -> Y
ex_intros (~ (forall x : A, P x)) DeMorgan050
: (exists x : A, P x -> False) -> ~ (forall x : A, P x)
```

最後にドモルガンの法則は, `ex_intros` に `DeMorgan050` を適用して証明される.

Theorem *DeMorgan05*: $(\exists x, \neg (P\ x)) \rightarrow \neg (\forall x, (P\ x))$.

Proof.

`apply (ex_intros _ DeMorgan051).`

Qed.

Print DeMorgan05.

```
DeMorgan05 =  
ex_intros (~ (forall x : A, P x)) DeMorgan050  
  : (exists x : A, ~ P x) -> ~ (forall x : A, P x)
```

2.3.2 $\neg(\exists x, P(x)) \rightarrow \forall x, \neg P(x)$

もうひとつのドモルガンの法則

Theorem 2.6 $\neg(\exists x, P(x)) \rightarrow \forall x, \neg P(x)$

を考えてみる.

$\neg(\exists x, P(x))$ の証明は関数 $(\exists x, P(x)) \rightarrow False$ であり, 関数族

$$\{f_a : P(a) \rightarrow False \mid a \in A\}$$

である. 目的とする関数は $x : A \rightarrow P(x) : Prop \rightarrow False$ であるので, まず, `ex_intros` の逆, 関数 $(\exists x, P(x))$ から Y への関数を関数族 $\{f_a : P(a) \rightarrow Y \mid a \in A\}$ へ分解する関数 `ex_case` を作る.

Lemma ex_case: $\forall Y:Prop, ((\exists x, (P x)) \rightarrow Y) \rightarrow (\forall x:A, ((P x) \rightarrow Y))$.

Proof.

move \Rightarrow *Y H a H1*.

apply *H*.

$\exists a$.

apply *H1*.

Qed.

Print ex_case.

```
ex_case =  
fun (Y : Prop) (H : (exists x : A, P x) -> Y) (a : A) (H1 : P a) =>  
H (ex_intro [eta P] a H1)  
  : forall Y : Prop, ((exists x : A, P x) -> Y) -> forall a : A, P a -> Y
```

そして, 次のドモルガンの法則が証明される.

Theorem DeMorgan06: $(\neg \exists x, P x) \rightarrow \forall x, \neg P x$.

Proof.

move \Rightarrow *H1 x Px*.

apply (ex_case False *H1 x*).

apply *Px*.

Qed.

Print DeMorgan06.

証明中, *H1*: $(\sim \text{exists } x, P x)$, *Px*: $P x$ に注意する.

```
DeMorgan06 =  
fun (H1 : ~ (exists x : A, P x)) (x : A) => [eta ex_case False H1 x]  
  : ~ (exists x : A, P x) -> forall x : A, ~ P x
```

2.3.3 $\forall x, \neg P(x) \rightarrow \neg(\exists x, P(x))$

次に Theorem 2.6 の逆

Theorem 2.7 $\forall x, \neg P(x) \rightarrow \neg(\exists x, P(x))$

を考えてみる. $\neg(\exists x, P(x))$ の証明は *Colimit* から *False* への関数であるので, `ex_intro` で実現可能である. 個別の証明から全体の証明を構築するように `ex_intro2` を構成する.

Lemma *ex_intros2*: $\forall Y:\text{Prop},$

$(\forall x, (P\ x) \rightarrow Y) \rightarrow ((\exists x, (P\ x)) \rightarrow Y).$

Proof.

`move \Rightarrow Y fx H1.`

`case H1.`

`move \Rightarrow x Px.`

`apply (fx x Px).`

Qed.

Theorem *DeMorgan07*: $(\forall x, \neg P\ x) \rightarrow (\neg \exists x, P\ x).$

Proof.

`move \Rightarrow H H1.`

`apply (ex_intros2 False H H1).`

Qed.

第3章 Library 03ElementaryTactics

3.1 第3回「証明の基本」

今回は, 最も基本的な命題「等式」と「否定」の証明についての基本的な証明の名前, また, タクティクの使い方をまとめる.

Require Import *Ssreflect.ssreflect Ssreflect.ssrbool Ssreflect.ssrfun List*.

3.2 等式の証明 (rewrite)

3.2.1 erefl

最も基本的な等式の証明を与える関数は, **erefl** である.

Lemma *opo*: $1 + 1 = 2$.

Proof.

compute.

apply (*erefl* 2).

Qed.

Print *opo*.

Print *erefl*.

等式の証明を与える関数は **erefl** : $A \rightarrow Prop$ である. 任意の型 A の元, $a \in A$ に対して, $a = a$ であることの証明を **erefl**(a) が与える. ここで $1 + 1$ の計算をコンピュータに任せると, 両辺がともに 2 なので, 証明は **erefl**(2) である.

```
opo = erefl 2
      : 1 + 1 = 2

Notation erefl := @eq_refl

Inductive eq (A : Type) (x : A) : A -> Prop := eq_refl : x = x

For eq: Argument A is implicit and maximally inserted
For eq_refl, when applied to no arguments:
  Arguments A, x are implicit and maximally inserted
For eq_refl, when applied to 1 argument:
  Argument A is implicit
For eq: Argument scopes are [type_scope _ _]
For eq_refl: Argument scopes are [type_scope _]
```

3.2.2 eq_ind

次に等式の証明で重要となるのが, `eq_ind` である.

```

eq_ind =
fun (A : Type) (x : A) => [eta eq_rect x]
  : forall (A : Type) (x : A) (P : A -> Prop),
    P x -> forall y : A, x = y -> P y

Argument A is implicit
Argument scopes are [type_scope _ _ _ _ _]

```

述語 $P(x)$ の証明を $F_x : P(x)$ を作ったとします. もし, 等式 $x = y$ が成り立つ, すなわち, 等式 $x = y$ の証明 ($H : x = y$) があれば, F_x と H から, $P(y)$ の証明を作る事が出来るはずです. $P(y)$ の証明を $F_y : P(y)$ と書くことにします. このとき, F_y を Coq の関数 `eq_ind` を使って,

$$F_y = (\text{eq_ind } x \ P \ F_x \ y \ H) : P(y)$$

と書きます.

ここで, $npo(x) = (x + 1 = 2)$ とすると, $npo(1) = (1 + 1 = 2)$, $npo(n) = (n + 1 = 2)$ となります. $npo(1)$ の証明は, $opo : npo(1)$ ですので, $1 = n$ の証明 ($H : 1 = n$) を仮定すれば,

$$(\text{eq_ind } 1 \ npo \ opo \ n \ H) : npo(n)$$

は, $npo(n)$ の証明になります.

Definition $npo\ (n:nat):Prop := (n + 1 = 2)$.

Lemma $npo2:\forall\ n:nat, (1 = n) \rightarrow (n + 1 = 2)$.

Proof.

`move => n H.`

`apply (eq_ind 1 npo opo n H).`

Qed.

Print npo .

Print $npo2$.

```

npo = fun n : nat => n + 1 = 2
      : nat -> Prop
npo2 =
fun n : nat => [eta eq_ind 1 np1 opo n]
      : forall n : nat, 1 = n -> n + 1 = 2

```

3.2.3 rewrite

等式の書き換えには, Coq では, `rewrite` タクティクスを使います. 次に `rewrite` を使った証明と, そこで構成された関数を示す.

Lemma $npo2r:\forall\ n:nat, (1 = n) \rightarrow (n + 1 = 2)$.

Proof.

move \Rightarrow n H .

rewrite $-H$.

compute.

apply (erefl 2).

Qed.

Print npo2r.

```
npo2r =
fun (n : nat) (H : 1 = n) =>
(fun _e0_ : 1 + 1 = 2 =>
eq_ind 1 (fun _pv_ : nat => _pv_ + 1 = 2) _e0_ n H)
(erefl 2)
: forall n : nat, 1 = n -> n + 1 = 2
```

3.3 構成と反対方向の性質 (inversion)

3.3.1 nat

Print nat.

```
Inductive nat : Set := 0 : nat | S : nat -> nat
```

構造体 `nat` の構成子 $S : nat \rightarrow nat$ は, $x : nat$ に対して, $S(x) : nat$. を保証する. すなわち, `nat` が帰納的加算集合 (recursively enumerable set) として定義されている.

次の補題の証明は, `f_equal` S で与えられる.

Lemma 3.1 $\forall x, \forall y, (x = y) \rightarrow (S(x) = S(y))$

Goal $\forall (x y : nat), (x = y) \rightarrow ((S\ x) = (S\ y))$.

move \Rightarrow $x\ y$.

apply (f_equal S).

Qed.

Print f_equal.

```
f_equal =
fun (A B : Type) (f : A -> B) (x y : A) (H : x = y) =>
match H in (_ = y0) return (f x = f y0) with
| erefl => erefl (f x)
end
: forall (A B : Type) (f : A -> B) (x y : A), x = y -> f x = f y
```

しかし, その逆, 次の補題の証明は自明ではない.

Lemma 3.2 $\forall x, \forall y, S(x) = S(y) \rightarrow (x = y)$

この補題を `f_equal` を用いて証明するために, $S(x)$ から, x を導く関数, `pre` を定義する.

```
Definition pre (x:nat):nat:=match x with
| 0 => 0
| (S n) => n
end.
```

H を $S(x) = S(y)$ の証明とすると, $(f_equal \text{ pre } H)$ は, $\text{pre}(S(x)) = \text{pre}(S(y))$, すなわち, $x = y$ の証明を与える.

Lemma ss: $\forall (x\ y:\text{nat}), (S\ x) = (S\ y) \rightarrow (x = y)$.

Proof.

`move => x y.`

`apply (f_equal pre).`

Qed.

Print ss.

```
ss =
fun x y : nat => [eta f_equal pre (y:=S y)]
      : forall x y : nat, S x = S y -> x = y
```

この証明のための, `pre` の作成などを自動で行ってくれるタクティクが, `inversion` である.

Lemma ssI: $\forall (x\ y:\text{nat}), (S\ x) = (S\ y) \rightarrow (x = y)$.

Proof.

`move => x y H.`

```
x : nat
y : nat
H : S x = S y
=====
x = y
```

`inversion H.`

```
x : nat
y : nat
H : S x = S y
H1 : x = y
=====
y = y
```

`apply (erefl y).`

Qed.

Print ssI.

```

ss1 =
fun (x y : nat) (H : S x = S y) =>
  (@^~ (erefl (S y)))
  match H in (_ = y0) return (y0 = S y -> x = y) with
  | erefl =>
    [eta fun H1 : S x = S y =>
      [eta fun H3 : x = y => eq_ind y (eq^~ y) (erefl y) x (eq_sym H3)]
      (f_equal (fun e : nat => match e with
        | 0 => x
        | S n => n
        end) H1)]
    end
  : forall x y : nat, S x = S y -> x = y

```

ここで, `eq_ind` が用いられているが, 今回の例では不要である. また, `eq^~` の記述の定義は以下の通り¹.

Notation "`f ^~ y`" := (fun x => f x y)

3.3.2 even

偶数の集合は次のように再帰的に定義される.

Inductive `even` : `nat` → **Prop** :=

| `even_0` : `even 0`

| `even_SS` : $\forall n, \text{even } n \rightarrow \text{even } (S (S n))$.

この定義は偶数の集合を帰納的可算集合で与えている. `even_0` と `even_SS` で, 偶数を列挙することは出来るが, 与えられた自然数 n を偶数かどうかを判断することは出来ない.

与えられた自然数を偶数かどうか判断するために, より小さな自然数に対する判定が行えるようにしたい. そこで, 与えられた自然数の2つ前の自然数を返す関数 `pre2` を作成する.

$$\text{pre2}(n) = \begin{cases} 0 & (n \leq 2) \\ n - 2 & (\text{otherwise}) \end{cases}$$

Definition `pre2` (`x:nat`):`nat`:=**match** `x` **with**

| `0` => `0`

| (`S n`) => **match** `n` **with**

| `0` => `0`

| (`S n1`) => `n1`

end

end.

`pre2` を用いて, 次の補題が証明される.

Lemma 3.3 $\forall x, \forall y, S(S(x)) = S(S(y)) \rightarrow (x = y)$

Lemma sss: $\forall (x y : \text{nat}), (S(S x)) = (S(S y)) \rightarrow (x = y)$.

Proof.

`move => x y.`

`apply (f_equal pre2).`

¹<http://ssr.msr-inria.inria.fr/doc/ssreflect-1.5/Ssreflect.ssrfun.html>

Qed.

次に補題

Lemma 3.4 $\forall n, \text{even}(n) \rightarrow \text{even}(\text{pre2}(n))$

を考える.

Lemma evenPre2: $\forall (n:\text{nat}), (\text{even } n) \rightarrow (\text{even } (\text{pre2 } n))$.

Proof.

move $\Rightarrow n$.

case.

compute.

apply *even_0*.

move $\Rightarrow n0\ H$.

compute.

apply *H*.

Qed.

Print evenPre2.

```
evenPre2 =
fun (n : nat) (_top_assumption_ : even n) =>
(fun (_evar_0_ : (fun (n0 : nat) (_ : even n0) => even (pre2 n0)) 0 even_0)
  (_evar_0_0 : forall (n0 : nat) (e : even n0),
    (fun (n1 : nat) (_ : even n1) => even (pre2 n1))
      (S (S n0)) (even_SS n0 e)) =>
  match
    _top_assumption_ as e in (even n0)
  return ((fun (n1 : nat) (_ : even n1) => even (pre2 n1)) n0 e)
with
| even_0 => _evar_0_
| even_SS x x0 => _evar_0_0 x x0
end) even_0 (fun n0 : nat => id)
  : forall n : nat, even n -> even (pre2 n)
```

そして, *evenPre2* を用いて次の補題が証明される.

Lemma 3.5 $\forall n, \text{even}(S(S(n))) \rightarrow \text{even}(n)$

Lemma evenSS: $\forall n, \text{even } (S(S\ n)) \rightarrow \text{even } n$.

move $\Rightarrow n$.

apply (*evenPre2* (*S* (*S* *n*))).

Qed.

3.4 否定の証明 (discriminate, inversion)

3.4.1 False

Check False_rec.

```
False_rec
  : forall P : Set, False -> P
```

False に向かう関数のひとつが `False_rec` である. この引数に *False* 自身を入れることで, *False* を得ることが出来る. では, その引数の *False* は, どうやって作るか? *False* そのものを仮定すれば, 当たり前だが, *False* へ向かう関数とその引数を仮定しても良い. $\neg P = P \rightarrow \text{False}$ であるから, *P* の証明 $p : P$ と $\neg P$ の証明 $np : (\neg P)$ から, *False* の証明 $np(p) : \text{False}$ を構成出来る. そして, `False_rec` を用いて任意の命題 *Q* が成立するという次の補題が証明される.

Lemma 3.6

$$P \rightarrow (\neg P) \rightarrow Q.$$

Lemma *contradict* : $\forall (P\ Q : \text{Prop}), P \rightarrow \neg P \rightarrow Q$.

Proof.

`move \Rightarrow P Q p np.`

`apply (False_rec Q (np p)).`

Qed.

3.4.2 $x \neq y$ (discriminate)

Check *eq_ind*.

$(x = y) \rightarrow \text{False}$ すなわち, $x \neq y$ の証明は, `eq_ind` を使って与える.

```
eq_ind
  : forall (A : Type) (x : A) (P : A -> Prop),
    P x -> forall y : A, x = y -> P y
```

`eq_ind` の定義より,

`(@eq_ind A x P H y)`

は, $((x = y) \rightarrow (P\ y))$ となる. ここで, $H : (P\ x)$ であることに注意する. すなわち, $(P\ x) = \text{True}$, $(P\ y) = \text{False}$ と計算出来る関数 *P* については, $I : \text{True}$ であるので,

`(@eq_ind A x P I y)`

は, $((x = y) \rightarrow \text{False})$ を与える. すなわち, 等式 $(x = y)$ の左辺の元に対して真, 右辺の元に対して偽となる命題 *P* を作成出来れば, それと `eq_ind` で $((x = y) \rightarrow \text{False})$ の証明を構成可能である.

$(1 \neq 0)$ を考えるとき, 自然数 `nat` 上の述語で, 値が 1 のときのみ *True* を返す述語 `eqone` を定義する.

Definition *eqone* (*x*:`nat`):`Prop`:=`match x with`

`| 0 \Rightarrow False`

`| S n \Rightarrow match n with`

`| 0 \Rightarrow True`

`| S _ \Rightarrow False`

`end`

end.

Compute `((eqone 0)::(eqone 1)::(eqone 2)::(eqone 3)::nil).`

```
Compute ((eqone 0)::(eqone 1)::(eqone 2)::(eqone 3)::nil).
= False :: True :: False :: False :: nil
: list Prop
```

(eqone 0)=False, (eqone 1)=True であることに注意する. このとき

(@eq_ind nat 1 eqone I 0)

が求める証明である.

Definition eqoneq:((1=0) → (eqone 0))

:=(@eq_ind nat 1 eqone I 0).

Check eqoneq.

具体的には上記の証明 eqoneq を用いて 1 0 が以下のように証明される.

Goal (1<>0).

Proof.

move ⇒ H.

apply (eqoneq H).

Qed.

誤った等式ごとに矛盾を導く関数 (eqone や eqoneq など) を eqind を用いて構成するのは面倒である. Coq では, discriminate タクティクがその関数を構成してくれる. 次に, discriminate を使った証明と, そこで構成された関数を示す.

Lemma onezero:(1<>0).

Proof.

move ⇒ H.

discriminate.

Qed.

Print onezero.

```
onezero =
fun H : 1 = 0 =>
[eta [eta False_ind False]]
  (eq_ind 1 (fun e : nat => match e with
                           | 0 => False
                           | S _ => True
                           end) I 0 H)
  : 1 <> 0
```

3.5 $\neg P(a)$ (inversion)

$\neg P(a)$ の証明は, $P(a) \rightarrow False$ の証明に対応する. $P(a)$ を直接計算して $False$ を導けない場合, $Q : A \rightarrow Prop$ で, $Q(a) = False$ が計算出来, かつ, $P(a) \rightarrow Q(a)$ が証明出来る述語 $Q(x)$ を考える. 具体的には, Q の補集合が有限集合であるものであれば $Q(a) = False$ は計算可能である. $H : P(a)$, $F : \forall x, P(x) \rightarrow Q(x)$, $Q(a) = False$ とすると, $F(a) : P(a) \rightarrow False$ となる.

$\neg(\text{even } 1)$ を考えよう. $\text{even} = \{x \mid (\text{even } x)\}$ よりも大きく, 決定可能な集合 $\text{notone} = \{x \mid x \neq 1\}$ を定義する. 偶数の集合 (even), および, 1 以外の集合 (notone) は次のように再帰的に定義される.

Definition notone (x:nat):Prop:=match x with

| 0 ⇒ True

```
| S n => match n with
  | 0 => False
  | (S _) => True
end
end.
```

証明すべきは, $\text{even} \subseteq \text{notone}$ であるが, notone と even の定義から, 帰納法で証明可能である.

Lemma *even_is_notone* ($x:\text{nat}$):(*even* x) \rightarrow (*notone* x).

Proof.

```
case.
rewrite /notone.
apply I.
move => n H.
rewrite /notone.
apply I.
```

Qed.

さて, $\text{even_is_notone} : \forall n, (\text{even } n) \rightarrow (\text{notone } n)$ より $(\text{even_is_notone } 1) : (\text{even } 1) \rightarrow (\text{notone } 1)$ である. また, $\neg(\text{even } 1) : (\text{even } 1) \rightarrow \text{False}$ であるが, $(\text{notone } 1) = \text{False}$ であるので, 命題が証明出来る.

Lemma *one_is_not_even*: $\neg (\text{even } 1)$.

Proof.

```
move => H.
apply (even_is_notone 1 H).
```

Qed.

Print *even_is_notone*.

Print *one_is_not_even*.

```
even_is_notone =
fun (x : nat) (_top_assumption_ : even x) =>
(fun (_evar_0_ : (fun (n : nat) (_ : even n) => notone n) 0 even_0)
  (_evar_0_0 : forall (n : nat) (e : even n),
    (fun (n0 : nat) (_ : even n0) => notone n0)
      (S (S n)) (even_SS n e)) =>
  match
    _top_assumption_ as e in (even n)
  return ((fun (n0 : nat) (_ : even n0) => notone n0) n e)
with
| even_0 => _evar_0_
| even_SS x0 x1 => _evar_0_0 x0 x1
end) I (fun (n : nat) (_ : even n) => I)
  : forall x : nat, even x -> notone x

one_is_not_even = [eta even_is_notone 1]
  : ~ even 1
```

この notone を探してくれるのが, inversion である.

Lemma *one_is_not_even_with_inversion*: $\neg (\text{even } 1)$.

Proof.

move \Rightarrow *H*.

inversion *H*.

Qed.

Print *one_is_not_even_with_inversion*.

```
one_is_not_even_with_inversion =
fun H : even 1 =>
  (@^~ (erefl 1))
  match H in (even n) return (n = 1 -> False) with
  | even_0 =>
    [eta fun H1 : 0 = 1 =>
      [eta [eta False_ind False]]
      (eq_ind 0
        (fun e : nat => match e with
          | 0 => True
          | S _ => False
        end) I 1 H1)]
  | even_SS n H0 =>
    (fun H2 : S (S n) = 1 =>
      [eta [eta False_ind (even n -> False)]]
      (eq_ind (S (S n))
        (fun e : nat =>
          match e with
          | 0 => False
          | 1 => False
          | S (S _) => True
        end) I 1 H2))^~ H0
    end
  : ~ even 1
```

文献 [?] の p.248 に "The Inner Workings of inversion" の記述がある.

第4章 Library 04TPPmark2014

4.1 第4回「TPPmark2014を解く」

今回は, TPP2014¹ で出題された以下の問題を解く.

Let $\mathbf{N} = \{0, 1, 2, 3, \dots\}$ be a set of natural numbers, $p \in \mathbf{N}$ and $q \in \mathbf{N}$. We denote $(p \bmod q) = r$ if and only if there exist $k \in \mathbf{N}$ and $r \in \mathbf{N}$ such that $p = kq + r$ and $0 \leq r < q$. Further, we denote $(q | p)$ if and only if $(p \bmod q) = 0$. Prove the following questions:

- (i) For any $a \in \mathbf{N}$, $(a^2 \bmod 3) = 0$ or $(a^2 \bmod 3) = 1$.
- (ii) Let $a \in \mathbf{N}$, $b \in \mathbf{N}$ and $c \in \mathbf{N}$. If $a^2 + b^2 = 3c^2$ then $(3 | a)$, $(3 | b)$ and $(3 | c)$.
- (iii) Let $a \in \mathbf{N}$, $b \in \mathbf{N}$ and $c \in \mathbf{N}$. If $a^2 + b^2 = 3c^2$ then $a = b = c = 0$.

Require Import *Ssreflect.ssreflect Ssreflect.eqtype Ssreflect.ssrfun Ssreflect.ssrbool Ssreflect.ssrnat MathComp.div MathComp.prime.*

4.2 補題

Lemma three *n P : P 0 → P 1 → P 2 → P (n % 3).*

Proof.

move ⇒ *P0 P1 P2*.

ltm_pmod : forall m d : nat, 0 < d -> m d < d

move: (*@ltm_pmod n 3*).

case (*n % 3*).

by [].

case.

by [].

case.

by [].

move ⇒ *n0*.

move/(*_ erefl*).

¹<http://imi.kyushu-u.ac.jp/lasm/tpp2014/>

※ `case (n %% 3)` 以降の繰り返しは, `[]` を使ってまとめて書ける.

- `|` 記号の左に最初のパートを右に次のパートの証明を書く.
- パートの中での `[]` はさらなる `case` での分割を表す.
- `[]` の前に `=>` があるので, 例えば `n0` は `move => n0` を意味する. また, `move /(_ erefl)` は `/(_ erefl)` だけ書けば良い.
- `move /(_ erefl)` は `move => H; move: (H erefl); move => {H}` と同じで, ゴールが $(X \rightarrow Y) \rightarrow Z$ のときの $(X \rightarrow Y)$ の部分を X の証明 `erefl` を使って Y に書き換える. 今の場合, $X=(0<3)$, $Y=(n0.+3<3)$, $Z=(P\ n0.+3)$ で, X の証明に `erefl` を使っている.
- `by []` は `//` と同じ.

`Restart` で最初から証明をやり直してみる.

Restart.

`move => P0 P1 P2; move: (@ltm_pmod n 3).`

`case : (n %% 3) => [|[]|[] n0 /(_ erefl)][] //.`

Qed.

Lemma nine n P : $P\ 0 \rightarrow P\ 1 \rightarrow P\ 2 \rightarrow P\ 3 \rightarrow P\ 4 \rightarrow P\ 5 \rightarrow P\ 6 \rightarrow P\ 7 \rightarrow P\ 8 \rightarrow P\ (n\ %%\ 9)$.

Proof.

`repeat (move => ?); move: (@ltm_pmod n 9).`

`case: (n %% 9) => [|[]|[]|[]|[]|[]|[]|[]|[]|[] n0 /(_ erefl)][] //.`

Qed.

Lemma dup a : $(a\ %%\ 9)\ %%\ 3 = a\ %%\ 3$.

Proof.

`modn_dvdm : forall m n d : nat, d %| m -> n %% m = n % [mod d]`

`by apply (@modn_dvdm 9 a 3).`

Qed.

Lemma 4.1 (divneq3) 自然数 x が $x \equiv 0 \pmod{3}$ を満たすとき,

$$x = 3 \left\lfloor \frac{x}{3} \right\rfloor.$$

Lemma divneq3 (x:nat): $(x\ %%\ 3 = 0) \rightarrow x = (x\ \% / 3) \times 3$.

Proof.

`move => H.`

`divn_eq : forall m d : nat, m = m % / d * d + m %% d`

`rewrite {1}(divn_eq x 3).`

`by [rewrite H].`

Qed.

Lemma 4.2 (inj9x) 自然数 x, y が $9x = 9y$ を満たすとき,

$$x = y.$$

Lemma inj9x ($x\ y:\text{nat}$): $x \times 9 = y \times 9 \rightarrow x = y$.

Proof.

```
introT : forall (P : Prop) (b : bool), reflect P b -> P -> b
elimT : forall (P : Prop) (b : bool), reflect P b -> b -> P
eqn_pm2l : forall m n1 n2 : nat, 0 < m -> (n1 * m == n2 * m) = (n1 == n2)
```

move $\Rightarrow H$; apply (introT eqP) in H.

※ move $\Rightarrow H$; apply (introT eqP) in H. は, move/eqP $\Rightarrow H$. と同じ.

```
rewrite (@eqn_pm2l 9 x y erefl) in H.
apply (elimT eqP) in H.
apply H.
```

※ apply (elimT eqP) in H. は, move/eqP in H. と同じ. move/eqP は等号==と=の両方向の書き換えに用いられる. Restart で最初からやり直し.

Restart.

move/eqP.

rewrite (@eqn_pm2l 9 x y erefl).

move/eqP.

by [].

Qed.

Lemma asqa0 ($a:\text{nat}$): $(a^2 = 0) \rightarrow (a = 0)$.

Proof.

by case a.

Qed.

Lemma asqa0m ($a:\text{nat}$): $(a^2 = 0 \% [mod\ 3]) \rightarrow (a = 0 \% [mod\ 3])$.

Proof.

```
modnXm : forall m n a : nat, (a %% n) ^ m = a ^ m % [mod n]
ltn_mod : forall m d : nat, (m %% d < d) = (0 < d)
```

rewrite -modnXm.

move: (@ltn_mod a 3).

case (a %% 3) \Rightarrow [[[[[n]]]] //.

Qed.

Lemma absqab0 ($a\ b:\text{nat}$): $(a^2 + b^2 = 0) \rightarrow (a = 0) \wedge (b = 0)$.

Proof.

move/eqP.

```
addn_eq0 : forall m n : nat, (m + n == 0) = (m == 0) && (n == 0)
```

rewrite *addn_eq0*.

move/*andP*.

elim.

move/*eqP* \Rightarrow *a2*.

move/*eqP* \Rightarrow *b2*.

apply (*conj* (*asqa0 a a2*) (*asqa0 b b2*)).

Qed.

Lemma *absqab0m* (*a b:nat*): ($a^2 + b^2 = 0 \text{ \% [mod 3]}$) \rightarrow ($a = 0 \text{ \% [mod 3]}$) \wedge ($b = 0 \text{ \% [mod 3]}$).

Proof.

```
modnDm : forall m n d : nat, m %% d + n %% d = m + n % [mod d]
```

rewrite $-(\text{modnDm } (a^2) (b^2)) -(\text{modnXm } 2 \ 3 \ a) -(\text{modnXm } 2 \ 3 \ b)$.

この時点でのゴールは

$$(a \% 3)^2 \% 3 + (b \% 3)^2 \% 3 = 0 \% [\text{mod } 3] \rightarrow a = 0 \% [\text{mod } 3] \wedge b = 0 \% [\text{mod } 3]$$

である。ここで、 $a \% 3$ と $b \% 3$ の部分に 0,1,2 の全ての値を入れて具体的に計算で論証するのが次の 1 行である。

apply (*three a*).

apply (*three b*).

by compute.

by apply (*three a*); apply (*three b*); compute.

Qed.

Lemma 4.3 (PropPmod3ab) 自然数 a, b, c が $a^2 + b^2 = 3c^2$ を満たすとき,

$$(a \equiv 0 \pmod{3}) \wedge (b \equiv 0 \pmod{3}).$$

Lemma *PropPmod3ab* (*a b c:nat*): ($a^2 + b^2 = 3 \times c^2$) \rightarrow ($a = 0 \text{ \% [mod 3]}$) \wedge ($b = 0 \text{ \% [mod 3]}$).

Proof.

move \Rightarrow *H*.

apply *absqab0m*.

rewrite *H*.

```
modnMr : forall p d : nat, (d * p) %% d = 0
```

by [rewrite (*modnMr* (c^2) 3)].

Qed.

Lemma *lemmal* (*a:nat*): ($a = 0 \text{ \% [mod 3]}$) \rightarrow ($a^2 = 0 \text{ \% [mod 9]}$).

Proof.

rewrite $-(\text{dup } a) -(\text{modnXm } 2 \ 9 \ a)$.

move: (*@ltm_mod a 9*).

by apply (*nine a*);compute.

Qed.

Lemma lemma2 (*a b:nat*): (*a* = 0 %[mod 3]) → (*b* = 0 %[mod 3]) → (*a* ^ 2) + (*b* ^ 2) = 0 %[mod 9].

Proof.

move ⇒ *Ha Hb*.

rewrite -(*modnDm* (*a*^2) (*b*^2)).

rewrite (*lemma1 a Ha*) (*lemma1 b Hb*).

by [compute].

Qed.

Lemma lemma3 (*c:nat*): 3*c = 0 %[mod 9] → *c* = 0 %[mod 3].

Proof.

modnMm : forall m n d : nat, m %% d * (n %% d) = m * n % [mod d]

rewrite -(*dup c*) -*modnMm*.

move: (@*ltm_mod c* 9).

by apply (*nine c*); compute.

Qed.

Lemma PropPmod3csq (*a b c:nat*): (*a* ^ 2) + (*b* ^ 2) = 3 × (*c* ^ 2) → (*c* ^ 2) = 0 %[mod 3].

Proof.

move ⇒ *H*.

move: (*PropPmod3ab a b c H*).

elim ⇒ *Ha Hb*.

move: (*lemma2 a b Ha Hb*).

rewrite *H* ⇒ *Hc*.

apply (*lemma3* (*c*^2) *Hc*).

Qed.

Lemma 4.4 (PropPmod3c) 自然数 *a, b, c* が $a^2 + b^2 = 3c^2$ を満たすとき,

$$(c \equiv 0 \pmod{3}).$$

Lemma PropPmod3c (*a b c:nat*): (*a* ^ 2) + (*b* ^ 2) = 3 × (*c* ^ 2) → *c* = 0 %[mod 3].

Proof.

move ⇒ *H*.

apply (*asqa0m c* (*PropPmod3csq a b c H*)).

Qed.

Lemma 4.5 (mod3lt) 自然数 *n* に対して,

$$\left\lfloor \frac{n+1}{3} \right\rfloor \leq n.$$

Lemma mod3lt (*n:nat*): (*n* + 1 %/ 3) ≤ *n*.

Proof.

```
ltm_Pdiv : forall m d : nat, 1 < d -> 0 < m -> m %/ d < m
```

by [apply (@ltm_Pdiv (n.+1) 3)].

Qed.

4.3 帰納法の復習

自然数 (nat) 上の証明は帰納的に定義された型 `nat` を定義した際に作られる証明 `nat_ind` を用いて行います. ここでは, 自然数上の数学的帰納法や超限帰納法に関わる補題を 3 つ証明します. 実際には, これらの補題を明示的に利用することなく, `Ssreflect` の Tactic である `elim` のオプション指定で超限帰納法での証明も直接行えます.

```
nat_ind
  : forall P : nat -> Prop,
    P 0 -> (forall n : nat, P n -> P n.+1) -> forall n : nat, P n
```

4.3.1 通常の帰納法

Check `nat_ind`.

Lemma 4.6 (induct0) 自然数上の命題 $P(n)$ に対して,
次の 2 つの命題が成立するとき, 任意の自然数 n に対して $P(n)$ が成り立つ.

- $P(0)$
- $\forall k, (P(k) \rightarrow P(k+1))$

Lemma `induct0 (n:nat) (P: nat → Prop): (P 0) → (∀ k:nat, (P k) → (P (k.+1))) → (P n).`

Proof.

move ⇒ `P0 Pk`.

apply (`nat_ind P P0 Pk`).

※ `elim n; [apply P0|apply Pk]`. と同じ.

Undo 1.

`elim n; [apply P0|apply Pk]`.

Qed.

Print `induct0`.

```

induct0 =
fun (n : nat) (P : nat -> Prop) (P0 : P 0) => (nat_ind P P0)^~ n
  : forall (n : nat) (P : nat -> Prop),
    P 0 -> (forall k : nat, P k -> P k.+1) -> P n
nat_ind =
[eta nat_rect]
  : forall P : nat -> Prop,
    P 0 -> (forall n : nat, P n -> P n.+1) -> forall n : nat, P n
nat_rect =
fun (P : nat -> Type) (f : P 0) (f0 : forall n : nat, P n -> P n.+1) =>
fix F (n : nat) : P n :=
  match n as n0 return (P n0) with
  | 0 => f
  | n0.+1 => f0 n0 (F n0)
end
  : forall P : nat -> Type,
    P 0 -> (forall n : nat, P n -> P n.+1) -> forall n : nat, P n

```

4.3.2 超限帰納法 (1)

Definition *transfinite* ($P:nat \rightarrow Prop$) ($n:nat$) := $\forall (x : nat), (x \leq n) \rightarrow (P x)$.

Lemma *transfinite0* ($P:nat \rightarrow Prop$): $(P 0) \rightarrow (transfinite P 0)$.

Proof.

move $\Rightarrow P0\ n0$.

```

leqn0      : forall n : nat, (n <= 0) = (n == 0)

```

rewrite *leqn0*.

move/*eqP* $\Rightarrow H$.

rewrite *H*.

apply *P0*.

上記は、次の1行にまとめられる。

Restart.

by move \Rightarrow ? ?; rewrite *leqn0* \Rightarrow /*eqP* \rightarrow .

Qed.

Lemma *transfiniten* ($P:nat \rightarrow Prop$) ($n:nat$): $(transfinite P n) \rightarrow (P n)$.

Proof.

move $\Rightarrow H$.

```

leqnn      : forall n : nat, n <= n

```

apply (*H* *n* (*leqnn* *n*)).

Qed.

Lemma *transfiniten1* (*P*:*nat* → **Prop**) (*n*:*nat*):
 (*transfinite P n*) → (*P n*.+1) → (*transfinite P n*.+1).

Proof.

move ⇒ *H1 Pn1 n1*.

leq_eqVlt : forall m n : nat, (m <= n) = (m == n) || (m < n)

rewrite *leq_eqVlt*.

move/*orP*.

case; [by move/*eqP* ->|apply (*H1 n1*)].

Qed.

Lemma 4.7 (induct1) 自然数上の命題 $P(n)$ に対して,
 次の 2 つの命題が成立するとき, 任意の自然数 n に対して $P(n)$ が成り立つ.

- $P(0)$
- $\forall k, ((P(l), (\forall l \leq k)) \rightarrow (P(l), (\forall l \leq (k + 1))))$

Lemma *induct1* (*n*:*nat*) (*P*: *nat* → **Prop**):

(*P 0*) → ($\forall k$:*nat*, (*transfinite P k*) → (*transfinite P (k*.+1))) → (*P n*).

Proof.

move ⇒ *P0 Pk*.

apply *transfiniten*.

move: *Pk*.

move: (*transfinite0 P P0*).

apply (*induct0 n (transfinite P)*).

※ 命題 (*transfinite P*) に関する帰納法はタクティク *elim* で *elim*: $n \{-2\}n$ (*leqnn n*) として始めることができる.

Restart.

move ⇒ *P0 Pk*.

elim: $n \{-2\}n$ (*leqnn n*); [apply (*transfinite0 P P0*)|apply *Pk*].

Qed.

Print *induct1*.

```
induct1 =
fun (n : nat) (P : nat -> Prop) (P0 : P 0)
  (Pk : forall k : nat, transfinite P k -> transfinite P k.+1) =>
transfiniten P n (induct1x n P (transfinite0 P P0) Pk)
  : forall (n : nat) (P : nat -> Prop),
    P 0 -> (forall k : nat, transfinite P k -> transfinite P k.+1) -> P n
```


4.3.3 超限帰納法 (2)

Lemma 4.8 (induct2) 自然数上の命題 $P(n)$ に対して,
次の2つの命題が成立するとき, 任意の自然数 n に対して $P(n)$ が成り立つ.

- $P(0)$
- $\forall k, (P(l), (\forall l \leq k)) \rightarrow P(k+1)$

Lemma induct2 $(n:\text{nat}) (P: \text{nat} \rightarrow \text{Prop})$:

$(P\ 0) \rightarrow (\forall k:\text{nat}, (\text{transfinite } P\ k) \rightarrow (P\ k.+1)) \rightarrow (P\ n).$

Proof.

move $\Rightarrow P\ 0\ H1$.

apply (induct1 $n\ P\ P0$).

move $\Rightarrow k\ Pk$.

apply (transfiniten1 $P\ k\ Pk\ (H1\ k\ Pk)$).

Qed.

4.4 命題

Definition defP $(a\ b\ c:\text{nat}) := a^2 + b^2 = 3 \times c^2$.

Definition defQ $(a\ b\ c:\text{nat}) := (a \% 3 = 0) \wedge (b \% 3 = 0) \wedge (c \% 3 = 0)$.

Theorem 4.1 (PropPtoQ) 自然数 a, b, c が $a^2 + b^2 = 3c^2$ を満たすとき,

$$(a \equiv 0 \pmod{3}) \wedge (b \equiv 0 \pmod{3}) \wedge (c \equiv 0 \pmod{3}).$$

Lemma PropPtoQ $(a\ b\ c:\text{nat}) : (\text{defP } a\ b\ c) \rightarrow (\text{defQ } a\ b\ c)$.

Proof.

rewrite /defP/defQ.

move $\Rightarrow H$.

rewrite -{2}(mod0n 3) -{5}(mod0n 3) -{8}(mod0n 3).

apply (conj (proj1 (PropPmod3ab $a\ b\ c\ H$))

(conj (proj2 (PropPmod3ab $a\ b\ c\ H$)) (PropPmod3c $a\ b\ c\ H$))).

Qed.

Theorem 4.2 (PropPmod3) 自然数 a, b, c が $a^2 + b^2 = 3c^2$ を満たすとき,

$$\left[\frac{a}{3}\right]^2 + \left[\frac{b}{3}\right]^2 = 3 \left[\frac{c}{3}\right]^2.$$

Lemma PropPmod3 $(a\ b\ c:\text{nat}) : (\text{defP } a\ b\ c) \rightarrow (\text{defP } (a \% 3) (b \% 3) (c \% 3)).$

Proof.

move $\Rightarrow H1$.

move: (PropPtoQ $a\ b\ c\ H1$).

elim $\Rightarrow H2$.

```
elim => H3 H4.
rewrite (divneq3 a H2) (divneq3 b H3) (divneq3 c H4) in H1.
rewrite /defP in H1.
```

```
expnMn : forall m1 m2 n : nat, (m1 * m2) ^ n = m1 ^ n * m2 ^ n
```

```
repeat rewrite expnMn in H1.
rewrite (.: (3^2)=9) in H1.
```

```
mulnDl : left_distributive muln addn
mulnA   : associative muln
```

```
rewrite -mulnDl in H1.
rewrite mulnA in H1.
apply (inj9x ((a %/ 3) ^ 2 + (b %/ 3) ^ 2) (3 × (c %/ 3) ^ 2) H1).
by [].
```

Qed.

Theorem 4.3 (defPc0) 自然数 a, b, c が $a^2 + b^2 = 3c^2$ を満たすとき,

$$c = 0.$$

Lemma PropPc0 $(a\ b\ c : nat) : (defP\ a\ b\ c) \rightarrow (c = 0).$

Proof.

```
move: a b.
apply (induct2 c) => [||k H1 a b PkI].
rewrite /transfinite in H1.
```

※ 超限帰納法を使う。 $c = 0$ の場合は自明。超限帰納法のための補題 `induct2` を準備していなくても、`elim: c {-2}c (leqnn c)` で超限帰納法を始めることが出来る。

```
move: (PropPmod3 a b (k.+1) PkI) => Pmod3.
move: (mod3lt k) => HkI.
move: (H1 (k.+1 %/ 3) HkI (a %/ 3) (b %/ 3) Pmod3) => Kdiv0.
move: (PropPmod3c a b k.+1 PkI) => Kmod0.
```

※ 補題 PropPmod3, 補題 mod3lt と既存の仮定を用いて, $\left\lfloor \frac{k+1}{3} \right\rfloor \leq k$ (Hk1) と $\left\lfloor \frac{a}{3} \right\rfloor^2 + \left\lfloor \frac{b}{3} \right\rfloor^2 = 3 \left\lfloor \frac{c}{3} \right\rfloor^2$ (Pmod3) が証明出来て仮定に加えることが出来る. そして, 帰納法の仮定 (H1) から $\left\lfloor \frac{k+1}{3} \right\rfloor = 0$ (Kdiv0) が求まる. $a^2 + b^2 = 3(k+1)^2$ (Pk1) と PropPmod3c から $(k+1) \equiv 0 \pmod{3}$ (Kmod0) が求まる. そして, 帰納過程の結論 $k+1 = 0$ を divn_eq を用いて導く.

```
H1 : forall x : nat, x <= k -> forall a0 b0 : nat, defP a0 b0 x -> x = 0
Pk1 : defP a b k.+1
Pmod3 : defP (a %/ 3) (b %/ 3) (k.+1 %/ 3)
Hk1 : k.+1 %/ 3 <= k
Kdiv0 : k.+1 %/ 3 = 0
Kmod0 : k.+1 = 0 % [mod 3]
Lemma divn_eq m d : m = m %/ d * d + m %% d.
```

by rewrite (divn_eq (k.+1) 3) Kdiv0 Kmod0.

Qed.

Theorem 4.4 (PropPab0) 自然数 a, b が $a^2 + b^2 = 0$ を満たすとき,

$$a = b = 0.$$

Lemma PropPab0 ($a b : \text{nat}$): ($\text{defP } a b 0$) $\rightarrow (a = 0) \wedge (b = 0)$.

Proof.

rewrite /defP.

rewrite (λx. 3 × 0 ^ 2 = 0).

apply absqab0.

by [compute].

Qed.

Theorem 4.5 (PropPabc0) 自然数 a, b, c が $a^2 + b^2 = 3c^2$ を満たすとき,

$$a = b = c = 0.$$

Lemma PropPabc0 ($a b c : \text{nat}$): ($\text{defP } a b c$) $\rightarrow (a = 0) \wedge (b = 0) \wedge (c = 0)$.

Proof.

move \Rightarrow Pabc.

move: (PropPc0 a b c Pabc) \Rightarrow C0.

rewrite C0 in Pabc.

move: (PropPab0 a b Pabc) \Rightarrow AB0.

apply (conj (proj1 AB0) (conj (proj2 AB0) C0)).

Qed.

謝辞

本節の内容は基本的には著者の責任でまとめていますが, TPPmark2014² 参加者の皆様の解答, および, TPP2014³ 参加者のみなさまのご助言に基づいています. ここにみなさまへの感謝の意を表します. ありがとうございました.

²<https://github.com/KyushuUniversityMathematics/TPP2014/wiki>

³<http://imi.kyushu-u.ac.jp/lasm/tpp2014/>