証明支援系 LEAN に入門しよう

梅崎直也@unaoya

第 6 回すうがく徒のつどい

目次

1. 証明とプログラムの関係
2. 述語論理と依存型
3. 証明を支援する仕組み
4. 微積分学の基本定理を証明する

今回の目標は、証明支援系の仕組みをなんとなく理解すること。使えるようになりたいかたは別のテキストなど参考文献をご覧ください。

資料について

今回使用するソースコード、スライドは以下の GitHub リポジトリにあります。

X のアカウント@unaoya 固定ポストにリンクがあります。

証明とプログラムの関係

はじめての証明

$1 = 1$ を証明する。

Intro.lean

```
def my_first_thm : 1 = 1 := Eq.refl 1
```

はじめての証明

$1 = 1$ を証明する。

Intro.lean

```
def my_first_thm : 1 = 1 := Eq.refl 1
```

Intro.lean

```
def my_first_thm : 1 = 1 := Eq.refl 1
```

LEAN の初歩

項 a が型 A を持つことを以下のように表す。

$$a : A$$

Intro.lean

```
def n : Nat := 1
def twice (n : Nat) : Nat := n + n
def twice' : Nat -> Nat :=
  fun n => n + n
def m : Nat := twice 3
def my_first_thm : 1 = 1 := Eq.refl 1
```

色々な型

自然数の型 Nat, 真偽値の型 Bool など。型 A, B から関数の型、直積型、直和型などを作れる。$f : A \to B$ で項 f が A から B への関数の型を持つことを表す。

$a \in A$ のようなものだと思える場合もあるし、ちょっとわかりにくい場合もある。

型も項になる。重要な型に $Prop$ がある。$P : Prop$ に対し、$h : P$ を「命題 P の証明 h」と解釈する。

命題が型、証明が項。命題 P を証明する、P 型の項を作る。命題 P を仮定する、P 型の項が与えられている。

証明とプログラム

証明がプログラムとはどういうことか。

証明とは？ おおよそ主張と理由の列と思える。理由となるのは論理規則または公理や定義またはすでに証明した命題。

プログラムとは？ 関数の組み合わせ、列および木構造。

プログラムの例

ある商店ではりんご、みかん、いちごの三種類の果物を売っている。1 個あたりの値段はりんご 100 円、みかん 50 円、いちご 200 円である。りんごの個数とみかんの個数といちごの個数を引数に取り、合計金額を返すプログラムを作ろう。ただし、レジ袋（1 枚だけ）が必要な場合は合計金額に 10 円を加える。

コーディング

Intro.lean

```
def total (a b c : Nat) (bag : Bool) : Nat :=
  a*100 + b*50 + c*200 + if bag then 10 else
```

$total : Nat \to Nat \to Nat \to Bool \to Nat$ という型を持つ関数を作った。$+ : Nat \to Nat \to Nat$ を使う。

プログラムの構造

関数とならば

P, Q を命題としたとき、$f : P \to Q$ は P の証明 $h : P$ を与えると Q の証明 $f(h) : Q$ を返す関数と解釈する。

プログラムは関数の組み合わせ、証明はならばの組み合わせ。関数は実際の値の対応関係が重要、証明は項の対応関係は気にしないで型だけ気にしてる。

証明は区別しておらず、関数があるかないかだけが重要

定理 A とその証明

a, b, c を自然数とし、$a = b$ と $b = c$ から $a = c$ を示す。

等式の推移律により、$a = b$ と $b = c$ から $a = c$ を導くことができる。

等号の推移律 $a = b$ と $b = c$ が仮定されると $a = c$ が導ける。（正確には a, b, c もパラメータ（引数）である。）$a = b$ の証明と $b = c$ の証明を受け取って $a = c$ の証明を返す関数である。（仮定の証明及び推移律がどういう関数であるかは気にしていない。）

定理 A の証明の構造

証明の構造

定理 A の証明のコーディング

等号の推移律を与える関数 $Eq.trans : a = b \to b = c \to a = c$ を用いる。これは $a = b$ 型の項と $b = c$ 型の項から $a = c$ 型の項を作る関数である。（正確には α, a, b, c も引数にとるがいまは省略）（依存型？ $a = b$ という型は a, b を変数（引数）にもつ関数 Eq により作られる型）

定理 B とその証明

a, b を自然数とする。$(a+1) \times b = a \times b + b$ が成り立つ。

$(a+1) \times b = a \times b + 1 \times b$ である。（分配法則）

$1 \times b = b$ である。（掛け算の定義）

$a \times b + 1 \times b = a \times b + b$ である。（関数 $x \mapsto a \times b + x$ と等号の定義）

$(a+1) \times b = a \times b + b$ である。（等号の推移律）

定理 B の証明の構造

木構造？ を明示する。証明の構造

定理 B の証明のコーディング

中間的な主張に名前をつける。(have を使うか let を使うか)

定理 C とその証明

a を自然数とする。$(a+1)\times(b+1)=a\times b+a+b+1$ が成り立つ。

一つ前の定理を使うと、
$(a+1)\times(b+1)=a\times(b+1)+(b+1)$ である。
$a\times(b+1)=a\times b+a$ である。

定理 C の証明の構造

証明の構造

定理 C の証明のコーディング

一つ前の定理を使う。

ここまでの整理

P を仮定すると Q が成り立つという定理。証明は P ならば P_1、P_1 ならば P_2、$\cdots$、P_{n-1} ならば Q という命題とその証明の列。

型 P から型 Q への関数を作りたい。関数 $P \to P_1, P_1 \to P_2, \ldots, P_{n-1} \to Q$（関数型の具体的な項）を合成することで $P \to Q$ を作ることができる。

違うところ。プログラムでは項の対応が重要な問題だが、定理の証明では型さえ合っていれば項の具体的な対応は気にしない。（証明同士を比較しない）

述語論理と依存型

直積/直和とかつ/または

関数 $P \times Q \to R$ を作る、関数 $P \to R$ か関数 $Q \to R$ を作ればよい。(それ以外にもありうる。)

関数 $R \to P \times Q$ を作る、関数 $R \to P$ と関数 $R \to Q$ を作る。

関数 $P + Q \to R$ を作る、関数 $P \to R$ と関数 $Q \to R$ を作る。

関数 $R \to P + Q$ を作る、関数 $R \to P$ か関数 $R \to Q$ を作ればよい。(それ以外にもありうる。)

述語論理

存在命題と全称命題。

依存型。$n = m$ という命題は変数 n, m に依存している。

$n = m$ は Prop 型の項であり、また $n = m$ 自体も型である。

依存和と依存積。族の直和と族の直積。

全称命題の証明例

実はすでに使ってた。関数と同一視している。

依存関数型？ def hoge (n : Nat) : P n := hoge 定義域は Nat で行き先の方は引数の値に依存して変わっている。定数族の直積が普通の関数。

全称命題のコーディング例

先ほどの定理は実は全称命題でした。

def hoge (a : A) := ba と def hoge : A -> B := fun a => ba は同じもの。

存在命題の証明例

自然数 n, m を考える。（ここに任意のを入れるかどうか）n が偶数であり、m がが偶数であるならば、$n + m$ は偶数である。

n が偶数であるので、その定義からある自然数 k が存在して、$n = 2k$ である。同様に、m が偶数であるので、ある自然数 l が存在して、$m = 2l$ である。$n + m = 2k + 2l = 2(k + l)$ であるから、$n + m$ は偶数である。

存在命題のコーディング例

コーディング

n が 4 の倍数ならば n は 2 の倍数である。n が 8 の倍数ならば n は 4 の倍数である。という二つの定理から、n が 8 の倍数ならば n は 2 の倍数である。を導く。

P ならば Q と Q ならば R から P ならば R を導く。関数の合成。

任意の n に対して、n が 4 の倍数ならば n は 2 の倍数である。

証明を支援する仕組み

証明の記述

面倒な部分は色々と便利な機能で省略していく。calc モードとか。タクティック、simp とか ring とか。これらを使うとどう簡略化できるかデモ。

項を直接書く（ここまで）

タクティックを使う。

これらの組み合わせもできる。

タクティック

タクティックについて。タクティックは関数ではなくモナド。関数は項を扱う。タクティックは項だけでなく状態を扱う。

タクティックで記述が簡単になる。ある程度の証明の自動化（simp を用いた自動化の例など）もできる。定理 A, B, C をタクティックを使って証明する。

ライブラリ

普通のプログラミング言語と同様、LEAN にもさまざまなライブラリが存在する。

普通は便利なデータ構造やそれらを扱う関数がライブラリにある。LEAN では数学的構造やそれらに関するの定理がライブラリにある。

ライブラリにあるかないかは、誰かが書いたかどうか、数学的な難しさなどはそこまで関係ない。（最近の論文の結果でも実装されているものもある、例示）

ライブラリの定理を使用。mathlib docs を見るとか。exact?とか。より便利なタクティックもライブラリにある。

ライブラリを使った証明

実は mathlib というライブラリには微積分学の基本定理がある。ライブラリを使った証明をまずは見てみよう。

微積分に関連する mathlib の定理の紹介。

微積分学の基本定理を証明する

目標の定理

$f : \mathbb{R} \to \mathbb{R}$ が連続であるとする。

$$\frac{d}{dx} \int_a^x f(t)dt = f(x)$$

今回はあえてライブラリを一切使わずに証明する。実数については公理だけを使う。（参考、杉浦解析入門）

極限の定義、関数の連続性の定義、微分の定義、積分の定義。

証明 ([杉浦] より引用)(1/4)

実数 x に対し、

$$F(x) = \int_a^x f(t)dt$$

とおく。(向きつき積分。f の連続性より、f は可積分)

任意の実数 x, y に対して等式

$$F(x) - F(y) = \int_y^x f(t)dt$$

が成り立ち、また三角不等式

$$\left| \int_y^x f(t)dt \right| \leq \left| \int_y^x |f(t)|dt \right|$$

が任意の x, y に対して成り立つことから、

証明 ([杉浦] より引用)(2/4)

任意の実数 $h \neq 0$ に対し、

$$\begin{aligned}\left|\frac{1}{h}(F(x+h)-F(x))-f(x)\right| &= \left|\frac{1}{h}\int_x^{x+h} f(t)dt - f(x)\right| \\ &\leq \left|\frac{1}{h}\int_x^{x+h} |f(t)-f(x)|dt\right|\end{aligned}$$

となる。

証明 ([杉浦] より引用)(3/4)

いま f は x で連続であるから、任意の $\epsilon > 0$ に対して、$\delta > 0$ が存在し、

$$|t - x| < \delta$$

ならば

$$|f(t) - f(x)| < \epsilon$$

となる。

証明 ([杉浦] より引用)(4/4)

そこで、$0 < |h| < \delta$ のとき上式の右辺は $\leq \epsilon$ となる。これは、

$$\lim_{h \neq 0, h \to 0} \frac{1}{h}(F(x+h) - F(x)) = f(x)$$

を意味する。すなわち F は x で微分可能で、$F'(x) = f(x)$ である。

定理の証明(1/3)

微分の定義から、

$$\frac{d}{dx}\int_a^x f(t)dt = \lim_{h\to 0}\frac{1}{h}\int_x^{x+h} f(t)dt$$

である。また、定数関数の積分から、任意の正の実数 h に対して、

$$f(x) = \frac{1}{h}\int_x^{x+h} f(x)dt$$

である。よって、

$$\lim_{h\to 0}\frac{1}{h}\int_x^{x+h} (f(x) - f(t))dt = 0$$

を言えば良い。

定理の証明(2/3)

$\lim_{h \to 0} \frac{1}{h} \int_x^{x+h} (f(x) - f(t))dt = 0$ を示す。

任意の $\epsilon > 0$ に対し、ある $\delta > 0$ が存在し、実数 h が $0 < |h| < \delta$ ならば、

$$\left| \frac{1}{h} \int_x^{x+h} (f(x) - f(t))dt \right| < \epsilon$$

であることを示したい。

$\epsilon > 0$ とする。f の連続性から、ある $\delta > 0$ が存在して、$|x - t| < \delta$ ならば $|f(x) - f(t)| < \epsilon$ である。このような δ を一つとり、これが条件を満たすことを示す。

定理の証明(3/3)

$\epsilon > 0$ とし、$\delta > 0$ が $|x - t| < \delta$ ならば $|f(x) - f(t)| < \epsilon$ であるとする。実数 h が $0 < |h| < \delta$ ならば、

$$\left| \frac{1}{h} \int_x^{x+h} f(x) - f(t) dt \right| < \epsilon$$

であることを示す。

$|t - x| \leq |h|$ ならば $|f(x) - f(t)| < \epsilon$ であり、積分の三角不等式、積分の単調性、定数関数の積分から、

$$\left| \int_x^{x+h} f(x) - f(t) dt \right| \leq \int_x^{x+h} |f(x) - f(t)| dt < \int_x^{x+h} \epsilon dt = \epsilon h$$

となる。

必要な積分の性質

三角不等式

単調性

定数関数の積分

コーディング(1/n)

証明に対応するコーディングをしよう。

参考資料

興味を持った人のために、参考資料、動画など