少人数クラス

中村　薫

2022 年 10 月 5 日

目 次

1　Coq による q-類似の形式化

1.1　Coq

Coq とは, 定理証明支援系の 1 つであり, 数学的な証明が正しいかどうか判定するプログラムである. 人間がチェックすることが難しい複雑な証明でも正しさが保証され, また証明付きプログラミングにも応用される. 例えば, 命題 P, Q について, $P \Longrightarrow Q$ かつ P であれば, Q が成り立つということは, Coq では

```
From mathcomp Require Import ssreflect.

Theorem modus_ponens (P Q : Prop) : (P → Q) ∧ P → Q.
Proof.
  move⇒ [] pq p.
  by apply pq.
Qed.
```

と表現できる.
Coq による証明は, Curry-Howard 同型と呼ばれる,

$$\text{命題} \leftrightarrow \text{型}$$
$$\text{証明} \leftrightarrow \text{型に要素が存在する}$$

という対応関係に基づいている. また, 各論理演算子について, 以下のように対応している

$$P \text{ ならば } Q \quad P \to Q$$
$$P \text{ かつ } Q \quad P \times Q$$
$$P \text{ または } Q \quad P + Q$$

この同型をもとに上記の証明をもう一度考えてみると, $P \to Q$ と P という型に要素が存在することから, Q という型の要素を構成すればよいということである.
まず, 前提の要素それぞれに pq, p と名前をつける. これがプログラム中の `move⇒ [] pq p` のことである. ここで, $P \to Q$ という型は, 入力する値の型が P, 出力する値の型が Q であるような関数の型であるため, P の要素 p に pq を適用することで, Q の要素を構成することができる. この関数適用がプログラム中の `apply pq` のことである.

1.2 q-類似

q-類似とは, $q \to 1$ とすると通常の数学に一致するような拡張のことである. 例えば, 自然数 n の q-類似 $[n]$ は

$$[n] = 1 + q + q^2 + \cdots q^{n-1}$$

であり, $(x-a)^n$ の q-類似 $(x-a)_q^n$ は

$$(x-a)_q^n := \begin{cases} 1 & (n = 0) \\ (x-a)(x-qa)\cdots(x-q^{n-1}a) & (n \geq 1) \end{cases}$$

である. 本章では, $D_q(x-a)_q^n = [n](x-a)_q^{n-1}$(ここで, D_q は微分の q-類似) が, n を整数に拡張しても成り立つことの形式化を目標とする.

1.3 形式化

様々な q-類似を考えるにあたって, まずは微分の q-類似から始める. 以下, q を 1 でない実数とする.

Def 1.3.1 ([1] p1 (1.1), p2 (1.5)) 関数 $f : \mathbb{R} \to \mathbb{R}$ に対して, $f(x)$ の q 差分 $d_q f(x)$ を,

$$d_q f(x) := f(qx) - f(x)$$

と定める. 更に, $f(x)$ の q 差分を $D_q f(x)$ を,

$$D_q f(x) := \frac{d_q f(x)}{d_q x} = \frac{f(qx) - f(x)}{(q-1)x}$$

と定める.

この定義を形式化すると,

```
From mathcomp Require Import all_ssreflect all_algebra.
Import GRing.

Section q_analogue.
Local Open Scope ring_scope.
Variable (R : rcfType) (q : R).
Hypothesis Hq : q - 1 ≠ 0.

Notation "f // g" := (fun x ⇒ f x / g x) (at level 49).

Definition dq (f : R → R) x := f (q * x) - f x.
Definition Dq f := dq f // dq id.
```

となる. このコードの意味は大まかに以下のとおりである.

- 最初の 2 行で必要なライブラリの指定をしている.
- `Variable` でそのセクション内で共通して使う変数を宣言している. `R` が Coq における実数の役割を果たす.
- `Hypothesis` で, q が 1 でないという仮定をしている. 使いやすさのため, $q \neq 1$ ではなく $q - 1 \neq 0$ という形にしている.
- `Notation` で関数同士の割り算の記法を定義している.
- 2 つの `Definition` で q-差分と q-微分をそれぞれ定義している. *coloneqq* 以前に定義の名前と引数, 以後に具体的な定義が書いてある. 例えば q-差分についてであれば, `d_q` が名前, `f` と `x` が引数, `f (q * x) - f x` が定義である. (`f` の後ろの`: R → R` は `f` の型である. 一方, もう一つの引数である `x` には型を書いていない. これは, Coq には強力な型推論があるため, 推論できるものであれば型を書く必要がないためである.) D_q の定義の中の `id` は恒等関数のことである.

Rmk 1.3.2 f が微分可能であるとき,

$$\lim_{q \to 1} D_q f(x) = \frac{d}{dx} f(x)$$

が成り立つが, 本稿においては極限操作に関しての形式化は扱わない.

次に, $x^n (n \in \mathbb{N})$ を q-微分した際にうまく振る舞うように自然数の q-類似を定義する.

Def 1.3.3 ([1] p2 (1.9)) $n \in \mathbb{N}$ に対して, n の q-類似 $[n]$ を,

$$[n] := \frac{q^n - 1}{q - 1}$$

と定義する.

```
Definition qnat n : R := (q ^ n - 1) / (q - 1).
```

この `qnat` に対して, $(x^n)' = nx^{n-1}$ の q-類似が成り立つ.

```
Lemma qderiv_of_pow n x :
  x ≠ 0 → Dq (fun x ⇒ x ^ n) x = qnat n * x ^ (n - 1).
Proof.
  move⇒ Hx.
  rewrite /Dq /dq /qnat.
  rewrite -{4}(mul1r x) -mulrBl expfzMl.
    rewrite -add_div.
    rewrite [in x ^ n](_ : n = (n -1) +1) //.
      rewrite expfzDr // expr1z.
      rewrite mulrA -mulNr !red_frac_r //.
      rewrite add_div //.
      rewrite -{2}[x ^ (n - 1)]mul1r.
      rewrite -mulrBl mulrC mulrA.
      by rewrite [in (q - 1)^-1 * (q ^ n - 1)] mulrC.
    by rewrite subrK.
  by apply mulf_neq0.
Qed.
```

[1] においては, この補題は Example([1] p2 (1.7)) として扱われ,

$$D_q x^n = \frac{(qx)^n - x^n}{(q-1)x} = \frac{q^n - 1}{q - 1} x^{n-1}$$

というように 1 行で終わっているが, 形式化する場合には何倍もかかっている. これは, 積の交換法則や分配法則といった「当たり前」の部分も明示的に書く必要があるからである. 証明中に多く出現している `rewrite` は等式の形をしている補題を使って証明すべき結論を書き換えるタクティックであり, *lem* が $A = B$ という形の補題のとき, `rewrite` *lem* で結論に現れる A を B に変える.
$a \in \mathbb{R}, n \in \mathbb{N}$ に対して, $(x-a)^n$ の q-類似は以下のように形式化できる.

```
Fixpoint qpoly_nonneg a n x :=
  match n with
  | 0 ⇒ 1
  | n.+1 ⇒ (qpoly_nonneg a n x) * (x - q ^ n * a)
  end.
```

[1] では,

$$(x-a)_q^n = \begin{cases} 1 & \text{if } n = 0 \\ (x-a)(x-qa)\cdots(x-q^{n-1}a) & \text{if } n \geq 1 \end{cases}$$

と定義している ([1] p8 Definition (3.4)) が, 形式化する際には `Fixpoint` を用いて, 再帰関数として定義している. この qpoly_nonneg について,

```
Theorem qderiv_qpoly_nonneg a n x :
  x ≠ 0 → Dq (qpoly_nonneg a n.+1) x = qnat n.+1 * qpoly_nonneg a n x.
Proof.
  move⇒ Hx.
  elim: n ⇒ [|n IH].
  - rewrite /Dq /dq /qpoly_nonneg /qnat.
    rewrite !mul1r mulr1 expr1z.
    rewrite opprB subrKA !divff //.
    by rewrite denom_is_nonzero.
  - rewrite (_ : Dq (qpoly_nonneg a n.+2) x =
                 Dq ((qpoly_nonneg a n.+1) **
                 (fun x ⇒ (x - q ^ (n.+1) * a))) x) //.
    rewrite qderiv_prod' //.
    rewrite [Dq (+%R^~ (- (q ^ n.+1 * a))) x] /Dq /dq.
    rewrite opprB subrKA divff //.
      rewrite mulr1 exprSz.
      rewrite -[q * q ^ n * a] mulrA -(mulrBr q) IH.
      rewrite -[q * (x - q ^ n * a) * (qnat n.+1 * qpoly_nonneg a n x)] mulrA.
      rewrite [(x - q ^ n * a) * (qnat n.+1 * qpoly_nonneg a n x)] mulrC.
      rewrite -[qnat n.+1 * qpoly_nonneg a n x * (x - q ^ n * a)] mulrA.
      rewrite (_ : qpoly_nonneg a n x * (x - q ^ n * a) = qpoly_nonneg a n.+1 x) //.
      rewrite mulrA.
      rewrite -{1}(mul1r (qpoly_nonneg a n.+1 x)).
      rewrite -mulrDl addrC.
      rewrite -(@divff _ (q - 1)) //.
      rewrite [qnat n.+1] /qnat.
      rewrite [q * ((q ^ n.+1 - 1) / (q - 1))] mulrA.
      rewrite (add_div _ _ (q -1)) //.
      by rewrite mulrBr -exprSz mulr1 subrKA.
    by apply denom_is_nonzero.
Qed.
```

2 HoTT

3 coqgen プロジェクト

参考文献

[1] Victor Kac, Pokman Cheung, *Quantum Calculus*, Springer, 2001.

[2] The Univalent Foundations Program, *Homotopy Type Theory: Univalent Foundations of Mathematics*