q-類似の Coq による形式化

アドバイザー： Jacquess Garrigue 教授
学籍番号： 322101289
氏名： 中村　薫

2022 年 12 月 21 日

目 次

1　序文

2　本文

2.1　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,

命題 ↔ 型
証明 ↔ 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, 論理演算子についても, 以下のような対応がある.

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

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.

2.2 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.

2.3 形式化

2.3.1 q-微分の定義

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.

Definition 2.3.1.1 ([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 (正確には `mathcomp` の `algebra` の実数) の役割を果たす. ここではまだ出てきていないが, `nat` が 0 を含む自然数に, `int` が整数に対応する.
- `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. := 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.

Remark 2.3.1.2 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 ($n \in \mathbb{Z}_{\geq 0}$) 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.

Definition 2.3.1.3 ([1] p2 (1.9)) $n \in \mathbb{Z}_{\geq 0}$ に対して, n の q-類似 $[n]$ を,

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

と定義する.

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

Proposition 2.3.1.4 ([1] p2 Example (1.7)) $n \in \mathbb{Z}_{>0}$ について,

$$D_q x^n = [n] x^{n-1}$$

が成り立つ.

Proof. 定義に従って計算すればよく,

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

□

この定義と補題の形式化は以下のとおりである.

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

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.
```

ここでも, コードについて少し説明を加える.

- `Definition` と同様, `Lemma` について, := の前に補題の名前と引数が, 後に補題の主張が書いてある. 今回であれば, `qderiv_of_pow` が補題の名前で, `n` と `x` が引数である.
- `Proof.` 以下が補題の証明である.
- *def* が定義のとき, `rewrite /`*def* で定義を展開している.
- *lem* が `A = B` という形の補題のとき, `rewrite` *lem* で結論に出現する `A` を `B` に書き換えている. 他のコマンドの使い方については [2] 等を参照.
- `red_frac_r` は,

  ```
  red_frac_r : ∀ x y z : R, z ≠ 0 → x * z / (y * z) = x / y
  ```

  という補題である. この補題を使うため, もともとはなかった $x \neq 0$ という前提を加えている. 実際, D_q の定義において分母に x が出現するので, x が 0 でないという前提は妥当である.

Remark 2.3.1.5 `qnat` という名前であるが, 実際には `n` の型は `nat` ではなく `R` にしている. また, `qderiv_of_pow` の `n` の型は `int` であるため, より一般化した形での形式化になっている.

[1] では証明は 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, 積の交換法則や指数法則などの, 通常の数学では「当たり前」なことが自動では計算されず, `rewrite mulrC` や `rewrite expfzDr` というように `rewrite` での書き換えを明示的に行わなければならないからである.

2.3.2 $(x-a)^n$ の q-類似

続いて $(x-a)^n$ の q-類似を定義し, その性質を調べる.

Definition 2.3.2.1 ([1] p8 Definition (3.4)) $x, a \in \mathbb{R}, n \in \mathbb{Z}_{\geq 0}$ に対して, $(x-a)^n$ の q-類似 $(x-a)_q^n$ を,

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

と定義する.

Proposition 2.3.2.2 $n \in \mathbb{Z}_{>0}$ に対し,

$$D_q(x-a)_q^n = [n](x-a)_q^{n-1}$$

が成り立つ.

Proof. n についての帰納法により示される. □

まず, $(x-a)_q^n$ の定義を形式化すると,

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

となる. `Fixpoint` を用いて再帰的な定義をしており, `match` を使って `n` が `0` かどうかで場合分けしている. 補題の証明については

```
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.
```

となる. ここで elim: n は n の帰納法に対応している.
指数法則については, 一般には $(x-a)^{m+n} \neq (x-a)_q^m (x-a)_q^n$ であり, 以下のようになる.

Proposition 2.3.2.3 ([1] p8 (3.6)) $x, a \in \mathbb{R}, m, n \in \mathbb{Z}_{>0}$ について,

$$(x-a)_q^{m+n} = (x-a)_q^m (x-q^m a)_q^n$$

が成り立つ.

Proof.

$$\begin{aligned}(x-a)_q^{m+n} &= (x-a)(x-qa)\cdots(x-q^{m-1}a) \times (x-q^m a)(x-q^{m+1}a)\cdots(x-q^{m+n-1}) \\ &= (x-a)(x-qa)\cdots(x-q^{m-1}a) \times (x-q^m a)(x-q(q^m x))\cdots(x-q^{n-1}(q^m a)) \\ &= (x-a)_q^m (x-q^m a)_q^n\end{aligned}$$

より成立する. □

この形式化は次のとおりである.

```
Lemma qpoly_nonneg_explaw x a m n :
  qpoly_nonneg a (m + n) x =
    qpoly_nonneg a m x * qpoly_nonneg (q ^ m * a) n x.
Proof.
  elim: n.
  - by rewrite addn0 /= mulr1.
  - elim ⇒ [_|n _ IH].
    + by rewrite addnS /= addn0 expr0z !mul1r.
    + rewrite addnS [LHS]/= IH /= !mulrA.
      by rewrite -[q ^ n.+1 * q ^ m] expfz_n0addr // addnC.
Qed.
```

[1] の証明では単に式変形しているが, qpoly_nonneg が再帰的に定義されているため, 形式化の証明では m, n に関する帰納法を用いている.
この指数法則を用いて, $(x-a)_q^n$ の n を負の数に拡張する. まず, [1] の定義は

Definition 2.3.2.4 ([1] p9 (3.7)) $x, a \in \mathbb{R}, l \in \mathbb{Z}_{>0}$ とする. このとき,

$$(x-a)_q^{-l} := \frac{1}{(x-q^{-l}a)_q^l}$$

と定める.

であり, この形式化は,

```
Definition qpoly_neg a n x := 1 / qpoly_nonneg (q ^ ((Negz n) + 1) * a) n x.
```

となる. ここで, Negz n とは Negz n = - n.+1 をみたすものであって, int は

```
Variant int : Set := Posz : nat → int | Negz : nat → int.
```

のように定義されている. よって, int は 0 以上か負かで場合分けできるため, n : int に対して,

```
Definition qpoly a n x :=
  match n with
  | Posz n0 ⇒ qpoly_nonneg a n0 x
  | Negz n0 ⇒ qpoly_neg a n0.+1 x
  end.
```

と定義できる.
整数に拡張した $(x-a)_q^n$ についても, 指数法則とと q-微分はうまく振る舞う. まず, 指数法則について,

Proposition 2.3.2.5 ([1] p10 Proposition 3.2) $m, n \in \mathbb{Z}$ について, Proposition 2.3.2.3 は成り立つ.

Proof. m, n の正負で場合分けして示す. $m > 0$ かつ $n > 0$ の場合はすでに示しており, $m = n = 0$ の場合は定義からすぐにわかる. その他の場合について, まず $m < 0$ かつ $n \geq 0$ の場合, $m = -m'$ とおくと

$$\begin{aligned}
(x-a)_q^m (x-q^m)_q^n &= (x-a)_q^{-m'} (x-q^{-m'}a)_q^n \\
&= \frac{(x-q^{-m'}a)_q^n}{(x-q^{-m'}a)_q^{m'}} \\
&= \begin{cases} (x-q^{m'}(q^{-m'}a))_q^{n-m'} & n \geq m' \\ \frac{1}{(x-q^n(q^{-m'}a))_q^{m'-n}} & n < m' \end{cases} \\
&= (x-a)_q^{n-m'} \\
&= (x-a)_q^{n+m}
\end{aligned}$$

というように, n と m' の大小で場合分けすることで示せる. 次に, $m \geq 0$ かつ $n < 0$ の場合, $n = -n'$ として,

$$\begin{aligned}
(x-a)_q^m (x-q^m)_q^n &= (x-a)_q^m (x-q^m a)_q^{-n'} \\
&= \begin{cases} \frac{(x-a)_q^{m-n'}(x-q^{m-n'}a)_q^{n'}}{(x-q^{m-n'}a)_q^{n'}} & m \geq n' \\ \frac{(x-a)_q^m}{(x-q^{m-n'})_q^{n'-m}(x-q^{n'-m}(q^{m-n'}a))} & m < n \end{cases} \\
&= \begin{cases} (x-a)^{m-n'} & m \geq n' \\ \frac{1}{(x-q^{m-n'})_q^{n'-m}} & m < n' \end{cases} \\
&= (x-a)_q^{m-n'} = (x-a)_q^{m+n}
\end{aligned}$$

となる. 最後に, $m < 0$ かつ $n < 0$ のとき, $m = -m'$, $n = -n'$ として,

$$\begin{aligned}
(x-a)_q^m (x-q^m)_q^n &= (x-a)_q^{-m'} (x-q^{-m'})_q^{-n'} \\
&= \frac{1}{(x-q^{-m'}a)_q^{m'}(x-q^{-n'-m'}a)_q^{n'}} \\
&= \frac{1}{(x-q^{-n'-m'}a)_q^{n'}(x-q^{n'}(q^{-m'-n'}a))_q^{m'}} \\
&= \frac{1}{(x-q^{-n'-m'}a)_q^{n'+m'}} \\
&= (x-a)_q^{-m'-n'} \\
&= (x-a)^{m'+n'}
\end{aligned}$$

となる. □

この補題を形式化すると,

```
Theorem qpoly_exp_law a m n x : q ≠ 0 →
  qpoly_denom a m x ≠ 0 →
  qpoly_denom (q ^ m * a) n x ≠ 0 →
  qpoly a (m + n) x = qpoly a m x * qpoly (q ^ m * a) n x.
Proof.
  move⇒ Hq0.
  case: m ⇒ m Hm.
  - case: n ⇒ n Hn.
    + by apply qpoly_nonneg_explaw.
    + rewrite qpoly_exp_pos_neg //.
      by rewrite addrC expfzDr // -mulrA.
  - case: n ⇒ n Hn.
    + by rewrite qpoly_exp_neg_pos.
    + by apply qpoly_exp_neg_neg.
Qed.
```

となる. 証明の構造としては, まず case:m で m が 0 以上か負かの場合分けを行い, 更にそれぞれの場合について case:n で n の場合分けを行っている. ここで, 前提の qpoly_denom の定義は

```
Definition qpoly_denom a n x := match n with
  | Posz n0 ⇒ 1
  | Negz n0 ⇒ qpoly_nonneg (q ^ Negz n0 * a) n0.+1 x
  end.
```

であり, 2 つの前提は補題の右辺に出現する項の分母が 0 にならないということである. 証明中に使われている補題のうち, qpoly_exp_pos_neg, qpoly_exp_neg_pos, qpoly_exp_neg_neg はそれぞれ $m \geq 0$ かつ $n < 0$, $m < 0$ かつ $n \geq 0$, $m < 0$ かつ $n < 0$ のときの証明の形式化であり, 例えば qpoly_exp_pos_neg については

```
Lemma qpoly_exp_pos_neg a (m n : nat) x : q ≠ 0 →
  qpoly_nonneg (q ^ (Posz m + Negz n) * a) n.+1 x ≠ 0 →
  qpoly a (Posz m + Negz n) x = qpoly a m x * qpoly (q ^ m * a) (Negz n) x.
Proof.
  move⇒ Hq0 Hqpolymn.
  case Hmn : (Posz m + Negz n) ⇒ [l|l]  /=.
  - rewrite /qpoly_neg mul1r.
    rewrite (_ : qpoly_nonneg a m x = qpoly_nonneg a (l + n.+1) x).
      rewrite qpoly_nonneg_explaw.
      have → : q ^ (Negz n.+1 + 1) * (q ^ m * a) = q ^ l * a.
        by rewrite mulrA -expfzDr // -addn1 Negz_addK addrC Hmn.
      rewrite -{2}(mul1r (qpoly_nonneg (q ^ l * a) n.+1 x)).
      rewrite red_frac_r.
        by rewrite divr1.
      by rewrite -Hmn.
    apply Negz_transp in Hmn.
    apply (eq_int_to_nat R) in Hmn.
    by rewrite Hmn.
  - rewrite /qpoly_neg.
    have Hmn' : n.+1 = (l.+1 + m)%N.
      move /Negz_transp /esym in Hmn.
      rewrite addrC in Hmn.
      move /Negz_transp /(eq_int_to_nat R) in Hmn.
      by rewrite addnC in Hmn.
    rewrite (_ : qpoly_nonneg (q ^ (Negz n.+1 + 1) * (q ^ m * a)) n.+1 x
              = qpoly_nonneg (q ^ (Negz n.+1 + 1) * (q ^ m * a))
                             (l.+1 + m) x).
      rewrite qpoly_nonneg_explaw.
      have → : q ^ (Negz n.+1 + 1) * (q ^ m * a) =
               q ^ (Negz l.+1 + 1) * a.
```

```
      by rewrite mulrA -expfzDr // !NegzS addrC Hmn.
    have → : q ^ l.+1 * (q ^ (Negz l.+1 + 1) * a) = a.
      by rewrite mulrA -expfzDr // NegzS NegzK expr0z mul1r.
    rewrite mulrA.
    rewrite [qpoly_nonneg (q ^ (Negz l.+1 + 1) * a) l.+1 x *
             qpoly_nonneg a m x] mulrC.
    rewrite red_frac_l //.
    have → : a = q ^ l.+1 * (q ^ (Posz m + Negz n) * a) ⇒ //.
      by rewrite mulrA -expfzDr // Hmn NegzK expr0z mul1r.
    apply qpoly_exp_non0r.
    rewrite -Hmn' //.
  by rewrite Hmn'.
Qed.
```

となっている. この証明についての注目点としては,

- [1] では m と n' の大小で場合分けをしていたが, 形式化では,

  ```
  case Hmn : (Posz m + Negz n) ⇒ [l|l]  /=.
  ```

  として, `m - n'` の値を `l` とおき, `l` が `0` 以上かどうかで場合分けをしている

- Coq では $A = B$ という等式はどの型の上でのものなのかが区別されている. `eq_int_to_nat` という補題は `int` 上の等式を `nat` 上の等式に写している.

などが挙げられる. さらに, q-微分については,

Proposition 2.3.2.6 ([1] p10 Proposition 3.3) $n \in \mathbb{Z}$ について,

$$D_q x^n = [n] x^{n-1}$$

が成り立つ. ただし, n が整数の場合にも, 自然数のときと同様, $[n]$ の定義は

$$\frac{q^n - 1}{q - 1}$$

である.

Proof. $n > 0$ のときは Proposition 2.3.2.2 であり, $n = 0$ のときは $[0] = 0$ からすぐにわかる. $n < 0$ のときは, Definition 2.3.2.4 と, 商の微分公式の q-類似版である

$$D_q\left(\frac{f(x)}{g(x)}\right) = \frac{g(x)D_q f(x) - f(x)D_q g(x)}{g(x)g(qx)} \quad \text{([1] p3 (1.13))}$$

及び Proposition 2.3.2.2 を用いて示される. □

[1] と同じ方針で証明する. まず, $n = 0$ のとき,

```
Lemma qderiv_qpoly_0 a x :
  Dq (qpoly a 0) x = qnat 0 * qpoly a (- 1) x.
Proof. by rewrite Dq_const qnat_0 mul0r. Qed.
```

である. ここで, `Dq_const` は

```
Lemma Dq_const x c : Dq (fun x ⇒ c) x = 0.
Proof. by rewrite /Dq /dq addrK' mul0r. Qed.
```

という定数関数の q-微分は 0 であるという補題である. 次に, $n < 0$ のときは

```
Lemma qderiv_qpoly_neg a n x : q ≠ 0 → x ≠ 0 →
  (x - q ^ (Negz n) * a) ≠ 0 →
  qpoly_nonneg (q ^ (Negz n + 1) * a) n x ≠ 0 →
  Dq (qpoly_neg a n) x = qnat (Negz n + 1) * qpoly_neg a (n.+1) x.
Proof.
  move⇒ Hq0 Hx Hqn Hqpoly.
  destruct n.
  - by rewrite /Dq /dq /qpoly_neg /= addrK' qnat_0 !mul0r.
  - rewrite qderiv_quot //.
      rewrite Dq_const mulr0 mul1r sub0r.
      rewrite qderiv_qpoly_nonneg // qpoly_qx // -mulNr.
      rewrite [qpoly_nonneg (q ^ (Negz n.+1 + 1) * a) n.+1 x *
                (q ^ n.+1 * qpoly_nonneg (q ^ (Negz n.+1 + 1 - 1) *
                  a) n.+1 x)] mulrC.
      rewrite -mulf_div.
      have → : qpoly_nonneg (q ^ (Negz n.+1 + 1) * a) n x /
                   qpoly_nonneg (q ^ (Negz n.+1 + 1) * a) n.+1 x =
                     1 / (x - q ^ (- 1) * a).
        rewrite -(mulr1
                    (qpoly_nonneg (q ^ (Negz n.+1 + 1) * a) n x)) /=.
        rewrite red_frac_l.
          rewrite NegzE mulrA -expfzDr // addrA -addn2.
          rewrite (_ : Posz (n + 2)%N = Posz n + 2) //.
          rewrite -{1}(add0r (Posz n)).
          by rewrite addrKA.
        by rewrite /=; apply mulnon0 in Hqpoly.
      rewrite mulf_div.
      rewrite -[q ^ n.+1 *
                 qpoly_nonneg (q ^ (Negz n.+1 + 1 - 1) * a) n.+1 x *
                   (x - q ^ (-1) * a)]mulrA.
      have → : qpoly_nonneg (q ^ (Negz n.+1 + 1 - 1) * a) n.+1 x *
                (x - q ^ (-1) * a) =
                qpoly_nonneg (q ^ (Negz (n.+1)) * a) n.+2 x ⇒ /=.
        have → : Negz n.+1 + 1 - 1 = Negz n.+1.
          by rewrite addrK.
        have → : q ^ n.+1 * (q ^ Negz n.+1 * a) = q ^ (-1) * a ⇒ //.
        rewrite mulrA -expfzDr // NegzE.
        have → : Posz n.+1 - Posz n.+2 = - 1 ⇒ //.
        rewrite -addn1 -[(n + 1).+1]addn1.
        rewrite (_ : Posz (n + 1)%N = Posz n + 1) //.
        rewrite (_ : Posz (n + 1 + 1)%N = Posz n + 1 + 1) //.
        rewrite -(add0r (Posz n + 1)).
        by rewrite addrKA.
      rewrite /qpoly_neg /=.
      rewrite (_ : Negz n.+2 + 1 = Negz n.+1) //.
      rewrite -mulf_div.
      congr (_ * _).
      rewrite NegzE mulrC.
      rewrite /qnat.
      rewrite -mulNr mulrA.
      congr (_ / _).
      rewrite opprB mulrBr mulr1 mulrC divff.
        rewrite invr_expz.
        rewrite (_ : - Posz n.+2 + 1 = - Posz n.+1) //.
        rewrite -addn1.
        rewrite (_ : Posz (n.+1 + 1)%N = Posz n.+1 + 1) //.
        rewrite addrC.
        rewrite [Posz n.+1 + 1]addrC.
```

```
        by rewrite -{1}(add0r 1) addrKA sub0r.
      by rewrite expnon0 //.
    rewrite qpoly_qx // mulf_neq0 //.
      by rewrite expnon0.
    rewrite qpoly_nonneg_head mulf_neq0 //.
    rewrite (_ : Negz n.+1 + 1 - 1 = Negz n.+1) //.
      by rewrite addrK.
    move: Hqpoly ⇒ /=.
    move/mulnon0.
    by rewrite addrK mulrA -{2}(expr1z q) -expfzDr.
Qed.
```

と, 非常に長くなっているが積の交換則や結合則などが多く, qderiv_quot が商の q-微分公式の形式化であるため, [1] の証明をそのまま形式化したものになっている. また, いくつかの項が 0 でないという条件がついているが, これらの項は Definition 2.3.2.4 において分母に現れるため, qderiv_of_pow のときと同様妥当であると考えられる. これらをまとめて,

```
Theorem qderiv_qpoly a n x : q ≠ 0 → x ≠ 0 →
  x - q ^ (n - 1) * a ≠ 0 →
  qpoly (q ^ n * a) (- n) x ≠ 0 →
  Dq (qpoly a n) x = qnat n * qpoly a (n - 1) x.
Proof.
  move⇒ Hq0 Hx Hxqa Hqpoly.
  case: n Hxqa Hqpoly ⇒ [|/=] n Hxqa Hqpoly.
  - destruct n.
    + by rewrite qderiv_qpoly_0.
    + rewrite qderiv_qpoly_nonneg //.
      rewrite (_ : Posz n.+1 - 1 = n) //.
      rewrite -addn1.
      rewrite (_ : Posz (n + 1)%N = Posz n + 1) //.
      by rewrite addrK.
  - rewrite Dq_qpoly_int_to_neg qderiv_qpoly_neg //.
        rewrite NegzK.
        rewrite (_ : (n + 1).+1 = (n + 0).+2) //.
        by rewrite addn0 addn1.
      rewrite (_ : Negz (n + 1) = Negz n - 1) //.
      by apply itransposition; rewrite NegzK.
    by rewrite NegzK addn1.
Qed.
```

と形式化できる. case: n で n が 0 以上か負かで場合分けを行い, destruct n で 0 か 1 以上かの場合分けをしており, それぞれの場合で qderiv_qpoly_0, qderiv_qpoly_nonneg, qderiv_qpoly_neg を使っていることが見て取れる.

2.3.3 q-Taylor 展開

この節では, 有限次 *Taylor* 展開の q-類似が成り立つこと, そしてその系として本論文の目的である Gauss's binomial formula が成り立つことを示し, 形式化する. まず, 一般に以下のことが成り立つことを確認しておく.

Theorem 2.3.3.1 ([1] p5 Theorem 2.1) $\mathbb{K} := \mathbb{R}$ または $\mathbb{C}$, $V := \mathbb{K}[x]$ とし, D を V 上の線型作用素とする. また, $\{P_n(x)\}_{n=0} \subset V$ $(n = 0, 1, 2, \cdots)$ は次の三条件をみたすとする.

$$\text{(i) } P_0 = 1,\ P_n(a) = 0 \quad ({}^\forall n \geq 1)$$
$$\text{(ii) } \deg P_n = n \quad ({}^\forall n \geq 0)$$
$$\text{(iii) } DP_n(x) = P_{n-1}(x) \quad ({}^\forall n \geq 1), \quad D(1) = 0$$

ただし, $a \in \mathbb{K}$ である. このとき, 任意の多項式 $f(x) \in V$ に対し, $\deg f(x) = N$ とすると,

$$f(x) = \sum_{n=0}^{N} (D^n f)(a) P_n(x)$$

が成り立つ.

この定理を形式化すると以下のようになる.

```
Theorem general_Taylor D n P (f : {poly R}) a :
  islinear D → isfderiv D P →
  (P 0%N).[a] = 1 →
  (∀ n, (P n.+1).[a] = 0) →
  (∀ m, size (P m) = m.+1) →
  size f = n.+1 →
  f = \sum_(0 ≤ i < n.+1) ((D \^ i) f).[a] *: P i.
```

記号の意味などは以下の通りである.

- {poly R} は R 係数多項式を表す型であり, p : poly R, a : R に対して p.[a] で多項式 p の a での値を, a *: p でスカラー倍を表す. また, size p は p の次数 +1 で定義されている.
- islinear, isfderiv はそれぞれ

  ```
  Definition islinear (D : {poly R} → {poly R}) :=
    ∀ a b f g, D ((a *: f) + (b *: g)) = a *: D f + b *: D g.

  Definition isfderiv D (P : nat → {poly R}) := ∀ n,
   match n with
    | 0 ⇒ (D (P n)) = 0
    | n.+1 ⇒ (D (P n.+1)) = P n
    end.
  ```

  という定義であり, 前者が線形作用素であること, 後者は条件 (iii) を形式化したものである.
- [1] での証明には, $\{P_0(x), P_1(x), \ldots, P_n(x)\}$ が V の基底となることを用いている. これを以下のように形式化した.

  ```
  Lemma poly_basis n (P : nat → {poly R}) (f : {poly R}) :
    (∀ m, size (P m) = m.+1) →
    (size f ≤ n.+1)%N →
    ∃ (c : nat → R), f = \sum_(0 ≤ i < n.+1) c i *: P i.
  ```

  実際には生成系であることを示している.

この定理において,

$$D \equiv D_q, \quad P_n \equiv \frac{(x-a)_q^n}{[n]!}$$

(ただし, $n \in \mathbb{Z}_{\geq 0}$ に対し, $[n]!$ を

$$[n]! := \begin{cases} 1 & (n = 0) \\ [n] \times [n-1] \times \cdots \times [1] & (n \geq 1) \end{cases}$$

と定める) とすることで, 有限次 Taylor 展開の q-類似が得られる.

Theorem 2.3.3.2 ([1] p12 Theorem 4.1) $f(x)$ を, N 次の実数係数多項式とする. 任意の $c \in \mathbb{R}$ に対し,

$$f(x) = \sum_{j=0}^{N} (D_q^j f)(c) \frac{(x-c)_q^j}{[j]!}$$

が成り立つ.

Proof. $\frac{(x-a)_q^n}{[n]!}$ が, a, D_q に対して Theorem 2.3.3.1 の三条件をみたすことを確かめればよい. (i), (ii) は $(x-a)_q^n$ の定義から, (iii) は Proposition2.3.2.2 から分かる. □

この定理を形式化したいが, 型が合わないという問題が発生する. 実際, `general_Taylor` の型を調べてみると,

```
general_Taylor
     : ∀ (D : {poly R} → {poly R}) (n : nat) (P : nat → {poly R}) (f : {poly R}) (a : R), ...
```

となっているが, `Dq`, `q_binom` の型は

```
Dq : (R → R) → R → R

q_binom_pos : R → nat → R → R
```

であり, 型が合わず代入できない. そこで, この問題を回避するため, 多項式に対しての q-微分と多項式としての $(x-a)_q^n$ を改めて定義する. まず, q-微分について,

```
Definition scaleq (p : {poly R}):= \poly_(i < size p) (q ^ i * p`_i).
Definition dqp p := scaleq p - p.
Definition Dqp p := dqp p %/ dqp 'X.
```

と定義する.

- `p`_i` は多項式 `p` の `i` 次の係数を表すため, `scaleq` は多項式 `p` を受け取り, i 次の係数を q^i 倍した多項式を返す操作である.
- もとの d_q の定義は $f(qx) - f(x)$ であるが, 多項式を入力して多項式を返す型で定義したいため, 値を入力することができないので, この形で定義した. 多項式に対しては d_q と同じ結果になることが確認できる (正確には, `dqp` を適用した多項式での `x` での値と, `x` ↦ `p.[x]` という関数に `dq` を適用した関数の `x` での値が等しいということである).

  ```
  Definition polyderiv (D : (R → R) → (R → R)) (p : {poly R}) :=
    D (fun (x : R) ⇒ p.[x]).
  Notation "D # p" := (polyderiv D p) (at level 49).
  Lemma dqp_dqE p x : (dqp p).[x] = (dq # p) x.
  ```

- `Dqp` の定義について, `p %/ p'` は多項式 `p` を多項式 `p'` で割った商を表しており, `'X` は x のみからなる単項式である. 実際に多項式に対して `Dqp` を計算すると, `dqp` の定義から, `dqp p` は定数項が打ち消しあい, また `dqp 'X` は `(q - 1) * 'X` となるので割り切れるはずである. 実際,

```
Lemma Dqp_ok p : dqp 'X %| dqp p.
```

が示せる (p' %| p で p が p' で割り切れることを表す).

今後は扱いやすさのため, 'X で約分した形

```
Definition Dqp' (p : {poly R}) := \poly_(i < size p) (qnat (i.+1) * p`_i.+1).
```

を用いる. このとき, Dqp と Dqp' が等しいことも示せる.

```
Lemma Dqp_Dqp'E p : Dqp p = Dqp' p.
```

この Lemma で注意すべき点は, X %/ 'X = 1%:P という計算をする際に特に条件が必要ないことである. Coq で約分の計算を行う際には分母が 0 でないという条件が必要だが, 'X は単項式であるため, ゼロ多項式とは異なる. よって自動的に条件がみたされることになる. 一方, Dqp と Dq が等しいことを示そうとすると,

```
Lemma Dqp'_DqE p x : x ≠ 0 → (Dqp' p).[x] = (Dq # p) x.
```

というように x != 0 という条件が必要になる. これは多項式の割り算ではなく実数の値の割り算での約分を計算する必要があるからである.
次に, $(x-a)_q^n$ を多項式として以下のように定義しなおす.

```
Fixpoint qbinom_pos_poly a n :=
  match n with
  | 0 ⇒ 1
  | n.+1 ⇒ (qbinom_pos_poly a n) * ('X - (q ^ n * a)%:P)
  end.
```

この多項式の x での値は元の定義の qbinom_pos と等しくなる.

```
Lemma qbinom_posE a n x :
  qbinom_pos a n x = (qbinom_pos_poly a n).[x].
```

参考文献

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

[2] https://www.math.nagoya-u.ac.jp/~garrigue/lecture/2021_AW/ssrcoq2.pdf