q-類似の Coq による形式化

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

2022 年 12 月 24 日

目 次

1 序文

本論文は, q-類似の初等的な結果を Coq によって形式化するものである. 具体的には [1] の 5 章前半, (5.5) 式

$$(x+a)_q^n = \sum_{j=0}^{n} \begin{bmatrix} n \\ j \end{bmatrix} q^{j(j-1)/2} a^j x^{n-j}$$

(ここで, $x, q \in \mathbb{R}, n \in \mathbb{Z}_{>0}$ であり, $(x+a)_q^n, \begin{bmatrix} n \\ j \end{bmatrix}$ はそれぞれ $(x+a)^n$, 二項係数の q-類似である) の形式化までを扱う. 本論文での q-類似に関する定義や定理, 証明は [1] によるものだが, その形式化を行ったという点において独自性がある基本的には, [1] での定義, 定理を述べた後, その形式化を与え, 必要であれば形式化をするにあたっての注意点を述べることを繰り返すという流れで構成していく. 証明の方針等は基本的に [1] の通りであるが, 2.3.3 では一部 [1] から離れ, 多項式として q-微分や q-二公式を定義しなおして形式化を行っている. これらの新たな定義が多項式に対してのもとの定義を適用したものと一致していることの証明も行っている.

q-類似の概要実数パラメータ q を' 極限で 1 に近づけると通常の数学に一致するような一般化自然数ならば $1, 1+q, 1+q+q^2$ が $1, 2, 3, \ldots$ に対応する. e^x, $sinx$ などの初等関数や積分の q-類似も考えられる　微分積分学の基本定理も成り立つ. ただしこれらは無限和が必要なので形式化はできていない. また, あえてパラメータを増やす q-類似を考える利点の一つとしては, 証明が複雑な定理に対してより簡単な別証明を与えられることである. 例えば, ヤコビの三重積 ([1] p35 Theorem 11.1)

$z, q \in \mathbb{R}, |q| < 1$ として,

$$\sum_{n=-\infty}^{\infty} q^{n^2} z^n = \prod_{n=1}^{\infty} (1-q^{2n})(1+q^{2n-1}z)(1+q^{2n-1}z^{-1})$$

が成り立つ.

はその一例である. 楕円関数論の文脈で登場する恒等式であるが (楕円関数論の教科書引用予定), q-類似で得られる式

$$(1+x)_q^{\infty} = \sum_{j=0}^{\infty} q^{j(j-1)/2} \frac{x^j}{(1-q)(1-q^2)\cdots(1-q^j)} \quad \text{([1] p30 (9.3) 式)}$$

$$\frac{1}{(1-x)_q^{\infty}} = \sum_{j=0}^{\infty} \frac{x^j}{(1-q)(1-q^2)\cdots(1-q^j)} \quad \text{([1] p30 (9.4) 式)}$$

を用いることで簡単に証明できる.

Coq の概要 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. 実際に Coq が用いられた有名な例として, 四色定理やフェイト・トンプソンの定理 (奇数位数定理) などがある. 今回の証明に関しては, Coq の標準ライブラリ ([4]) に加えて, 数学の証明のために整備されたライブラリである mathcomp([5]) も用いている. Coq や mathcomp の使い方については, [3] 等を参照のこと.

2 本文

2.1 Coq

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 $\leftrightarrow$ 型
証明 $\leftrightarrow$ 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.

P ならば Q	$P \to Q$
P かつ Q	$P \times Q$
P または Q	$P + Q$

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.

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. 本論文では, この $(x-a)_q^n$ に対して,

$$(x+a)_q^n = \sum_{j=0}^{n} \begin{bmatrix} n \\ j \end{bmatrix} q^{\frac{j(j-1)}{2}} a^j x^{n-j}$$

が成り立つことの形式化を目標としている (ただし, $\begin{bmatrix} n \\ j \end{bmatrix}$ は二項係数の q-類似). また, 本論文では扱わないが, q-類似を考える利点の一つとして,

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 40).

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,

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

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 Dq_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 -add_div; last first.
    by apply mulf_neq0.
  rewrite [in x ^ n](_ : n = (n -1) +1) //; last first.
    by rewrite subrK.
  rewrite expfzDr ?expr1z ?mulrA -?mulNr ?red_frac_r ?add_div //.
  rewrite -{2}[x ^ (n - 1)]mul1r -mulrBl mulrC mulrA.
  by rewrite [in (q - 1)^-1 * (q ^ n - 1)] mulrC.
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, `Dq_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, `Dq_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 qbinom_pos a n x :=
  match n with
  | 0 ⇒ 1
  | n.+1 ⇒ (qbinom_pos 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 Dq_qbinom_pos a n x : x ≠ 0 →
  Dq (qbinom_pos a n.+1) x =
  qnat n.+1 * qbinom_pos a n x.
Proof.
  move⇒ Hx.
  elim: n ⇒ [|n IH].
  - rewrite /Dq /dq /qbinom_pos /qnat.
    rewrite !mul1r mulr1 expr1z.
    rewrite opprB subrKA !divff //.
    by rewrite denom_is_nonzero.
  - rewrite (_ : Dq (qbinom_pos a n.+2) x =
                 Dq ((qbinom_pos a n.+1) **
                 (fun x ⇒ (x - q ^ (n.+1) * a))) x) //.
    rewrite Dq_prod' //.
    rewrite [Dq (+%R^~ (- (q ^ n.+1 * a))) x]/Dq /dq.
    rewrite opprB subrKA divff //; last first.
      by apply denom_is_nonzero.
    rewrite mulr1 exprSz.
    rewrite -[q * q ^ n * a]mulrA -(mulrBr q) IH.
    rewrite -[q * (x - q ^ n * a) * (qnat n.+1 * qbinom_pos a n x)]mulrA.
    rewrite [(x - q ^ n * a) * (qnat n.+1 * qbinom_pos a n x)]mulrC.
    rewrite -[qnat n.+1 * qbinom_pos a n x * (x - q ^ n * a)]mulrA.
    rewrite (_ : qbinom_pos a n x * (x - q ^ n * a) = qbinom_pos a n.+1 x) //.
    rewrite mulrA -{1}(mul1r (qbinom_pos a n.+1 x)).
    by rewrite -mulrDl -qnat_cat1.
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 qbinom_pos_explaw x a m n :
  qbinom_pos a (m + n) x =
    qbinom_pos a m x * qbinom_pos (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, qbinom_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 qbinom_neg a n x := 1 / qbinom_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 qbinom a n x :=
  match n with
  | Posz n0 ⇒ qbinom_pos a n0 x
  | Negz n0 ⇒ qbinom_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 qbinom_explaw a m n x : q ≠ 0 →
  qbinom_denom a m x ≠ 0 →
  qbinom_denom (q ^ m * a) n x ≠ 0 →
  qbinom a (m + n) x = qbinom a m x * qbinom (q ^ m * a) n x.
Proof.
  move⇒ Hq0.
  case: m ⇒ m Hm.
  - case: n ⇒ n Hn.
    + by apply qbinom_pos_explaw.
    + rewrite qbinom_exp_pos_neg //.
      by rewrite addrC expfzDr // -mulrA.
```

```
  - case: n ⇒ n Hn.
    + by rewrite qbinom_exp_neg_pos.
    + by apply qbinom_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 qbinom_denom の定義は

```
Definition qbinom_denom a n x := match n with
  | Posz n0 ⇒ 1
  | Negz n0 ⇒ qbinom_pos (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, qbinom_exp_pos_neg, qbinom_exp_neg_pos, qbinom_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 qbinom_exp_pos_neg については

```
Lemma qbinom_exp_pos_neg a (m n : nat) x : q ≠ 0 →
  qbinom_pos (q ^ (Posz m + Negz n) * a) n.+1 x ≠ 0 →
  qbinom a (Posz m + Negz n) x = qbinom a m x * qbinom (q ^ m * a) (Negz n) x.
Proof.
  move⇒ Hq0 Hqbinommn.
  case Hmn : (Posz m + Negz n) ⇒ [l|l]  /=.
  - rewrite /qbinom_neg mul1r.
    rewrite (_ : qbinom_pos a m x = qbinom_pos a (l + n.+1) x).
      rewrite qbinom_pos_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 (qbinom_pos (q ^ l * a) n.+1 x)) 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 /qbinom_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 (_ : qbinom_pos (q ^ (Negz n.+1 + 1) * (q ^ m * a)) n.+1 x
               = qbinom_pos (q ^ (Negz n.+1 + 1) * (q ^ m * a))
                            (l.+1 + m) x).
      rewrite qbinom_pos_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 [qbinom_pos (q ^ (Negz l.+1 + 1) * a) l.+1 x *
               qbinom_pos 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 qbinom_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) ⇒ [1|1]  /=.
  ```

  として, `m - n'` の値を `1` とおき, `1` 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 Dq_qbinomn0 a x :
  Dq (qbinom a 0) x = qnat 0 * qbinom a (- 1) x.
Proof. by rewrite Dq_const qnat0 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

```
Theorem Dq_qbinom_neg a n x : q ≠ 0 → x ≠ 0 →
  (x - q ^ (Negz n) * a) ≠ 0 →
  qbinom_pos (q ^ (Negz n + 1) * a) n x ≠ 0 →
  Dq (qbinom_neg a n) x = qnat (Negz n + 1) * qbinom_neg a (n.+1) x.
Proof.
  move⇒ Hq0 Hx Hqn Hqbinom.
  destruct n.
  - by rewrite /Dq /dq /qbinom_neg /= addrK' qnat0 !mul0r.
  - rewrite Dq_quot //.
      rewrite Dq_const mulr0 mul1r sub0r.
      rewrite Dq_qbinom_pos // qbinom_qx // -mulNr.
      rewrite [qbinom_pos (q ^ (Negz n.+1 + 1) * a) n.+1 x *
                (q ^ n.+1 * qbinom_pos (q ^ (Negz n.+1 + 1 - 1) *
```

```
            a) n.+1 x)] mulrC.
    rewrite -mulf_div.
    have → : qbinom_pos (q ^ (Negz n.+1 + 1) * a) n x /
                qbinom_pos (q ^ (Negz n.+1 + 1) * a) n.+1 x =
                  1 / (x - q ^ (- 1) * a).
      rewrite -(mulr1 (qbinom_pos (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) //.
        by rewrite -{1}(add0r (Posz n)) addrKA.
      by rewrite /=; apply mulnon0 in Hqbinom.
    rewrite mulf_div.
    rewrite -[q ^ n.+1 *
              qbinom_pos (q ^ (Negz n.+1 + 1 - 1) * a) n.+1 x *
                (x - q ^ (-1) * a)]mulrA.
    have → : qbinom_pos (q ^ (Negz n.+1 + 1 - 1) * a) n.+1 x *
             (x - q ^ (-1) * a) =
             qbinom_pos (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 /qbinom_neg /=.
    rewrite (_ : Negz n.+2 + 1 = Negz n.+1) // -mulf_div.
    congr (_ * _).
    rewrite NegzE mulrC /qnat -mulNr mulrA.
    congr (_ / _).
    rewrite opprB mulrBr mulr1 mulrC divff; last first.
      by rewrite expnon0.
    rewrite invr_expz (_ : - Posz n.+2 + 1 = - Posz n.+1) //.
    rewrite -addn1 (_ : Posz (n.+1 + 1)%N = Posz n.+1 + 1) //.
    by rewrite addrC [Posz n.+1 + 1]addrC -{1}(add0r 1) addrKA sub0r.
  rewrite qbinom_qx // mulf_neq0 //.
    by rewrite expnon0.
  rewrite qbinom_pos_head mulf_neq0 //.
  rewrite (_ : Negz n.+1 + 1 - 1 = Negz n.+1) //.
    by rewrite addrK.
  move: Hqbinom ⇒ /=.
  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, Dq_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, Dq_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 Dq_qbinom a n x : q ≠ 0 → x ≠ 0 →
  x - q ^ (n - 1) * a ≠ 0 →
  qbinom (q ^ n * a) (- n) x ≠ 0 →
  Dq (qbinom a n) x = qnat n * qbinom a (n - 1) x.
Proof.
  move⇒ Hq0 Hx Hxqa Hqbinom.
```

```
  case: n Hxqa Hqbinom ⇒ [|/=] n Hxqa Hqbinom.
  - destruct n.
    + by rewrite Dq_qbinomn0.
    + rewrite Dq_qbinom_pos //.
      rewrite (_ : Posz n.+1 - 1 = n) // -addn1.
      by rewrite (_ : Posz (n + 1)%N = Posz n + 1) ?addrK.
  - rewrite Dq_qbinom_int_to_neg Dq_qbinom_neg //.
        rewrite Negz_addK.
        rewrite (_ : (n + 1).+1 = (n + 0).+2) //.
        by rewrite addn0 addn1.
      rewrite (_ : Negz (n + 1) = Negz n - 1) //.
      by apply itransposition; rewrite Negz_addK.
    by rewrite Negz_addK 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 `Dq_qbinom_0`, `Dq_qbinom_nonneg`, `Dq_qbinom_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 scale_var (p : {poly R}):= \poly_(i < size p) (q ^ i * p`_i).
Definition dqp p := scale_var 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, scale_var 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 d_q 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.
```

Remark 2.3.3.3 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].
```

この Dqp と qbinom_pos_poly に対しても Proposition 2.3.2.2 と同じことが成り立つ.

```
Lemma Dqp'_qbinom_poly a n :
  Dqp' (qbinom_pos_poly a n.+1) = (qnat n.+1) *: (qbinom_pos_poly a n).
```

Remark 2.3.3.4 基本的な証明の方針は [1] と同じだが, Dq_prod' に対応する補題の証明のため, scale_var が積について分解できること, つまり

```
Lemma scale_var_prod (p p' : {poly R}) : scale_var (p * p') = scale_var p * scale_var p'.
```

を示しているが, 証明は

```
Proof.
  pose n := size p.
  have : (size p ≤ n)%N by [].
  clearbody n.
  have Hp0 : ∀ (p : {poly R}), size p = 0%N →
    scale_var (p * p') = scale_var p * scale_var p'.
    move⇒ p0.
    move/eqP.
    rewrite size_poly_eq0.
    move/eqP →.
    by rewrite mul0r scale_varC mul0r.
  elim: n p ⇒ [|n IH] p Hsize.
  ...
  Qed.
```

というように, 多項式の size に関する帰納法を使いたいため, (size p <= n)%N という仮定を加えている.

これで準備が整ったため, Theorem 2.3.3.2 を形式化する.

```
Fixpoint qfact n := match n with
  | 0 ⇒ 1
  | n.+1 ⇒ qfact n * qnat n.+1
  end.

Theorem q_Taylorp n (f : {poly R}) c :
  (∀ n, qfact n ≠ 0) →
  size f = n.+1 →
  f = \sum_(0 ≤ i < n.+1) ((Dqp' \^ i) f).[c] *: (qbinom_pos_poly c i / (qfact i)%:P).
```

Dqp, qbinom_pos_poly をもとの定義に戻したものについては,

```
Theorem q_Taylor n (f : {poly R}) x c :
  q ≠ 0 →
  c ≠ 0 →
  (∀ n, qfact n ≠ 0) →
  size f = n.+1 →
  f.[x] =  \sum_(0 ≤ i < n.+1)
             ((Dq \^ i) # f) c * qbinom_pos c i x / qfact i.
```

というように形式化できる.

Remark 2.3.3.5 c != 0 という条件は約分のためのものであるが, q != 0 はなぜ必要なのであろうか. これは, 高階 Dqp' と Dq を一致させる補題

```
Lemma hoDqp'_DqE p x n : q ≠ 0 → x ≠ 0 →
  ((Dqp' \^ n) p).[x] = ((Dq \^ n) # p) x.
Proof.
  move⇒ Hq0 Hx.
  rewrite /(_ # _).
  elim: n x Hx ⇒ [|n IH] x Hx //=.
  rewrite Dqp'_DqE // {2}/Dq /dq -!IH //.
  by apply mulf_neq0 ⇒ //.
Qed.
```

の証明において, `IH` を使う際に `q * x != 0` という条件が必要だからである. さらに, 高階 `Dq` が定義できないという点からも `q != 0` という条件は妥当である. 実際, $q = 0$ のとき,

$$\begin{aligned}
(D_q^2 f)(x) = (D_0^2 f)(x) &= (D_0(D_0 f))(x) \\
&= D_0\left(\lambda x.\ \frac{f(0x) - f(x)}{(0-1)x}\right)(x) \\
&= D_0\left(\lambda x.\ \frac{f(x) - f(0)}{x}\right)(x) \\
&= (D_0 F)(x) \quad (\text{ここで } F \coloneqq \lambda x.\ \frac{f(x) - f(0)}{x} \text{とおいた}) \\
&= \lambda x.\ \frac{F(x) - F(0)}{x}(x) \\
&= \frac{F(x) - F(0)}{x}
\end{aligned}$$

となるが,

$$F(0) = \frac{f(0) - f(0)}{0} = \frac{0}{0}$$

となってしまう (Coq では `0 / 0` は `0` と計算されるが, これでも正しい計算結果とはならない). この問題が起きるのは `dq` を値の代入を用いて定義しているからであり, 係数を変化させることで定義している `dqp` では `q = 0` でも問題が起きない. よってこの `dqp` を用いている `Dqp` および `Dqp'` については `q = 0` かどうかにかかわらず高階の操作を定義できる.

本論文の最後に, x^n と $(x-a)_q^n$ にこの Taylor 展開の q-類似を適用する.

Lemmma 2.3.3.6 ([1] p12 Example (4.4)) $n \in \mathbb{Z}_{>0}$ について,

$$x^n = \sum_{j=0}^{n} \begin{bmatrix} n \\ j \end{bmatrix} (x-1)_q^j \quad \left(\text{ここで, } \begin{bmatrix} n \\ j \end{bmatrix} \coloneqq \frac{[n]!}{[j]![n-j]!}\right)$$

が成り立つ.

Proof. Theorem 2.3.3.2 において, $f(x) = x^n$, $c = 1$ とする. 任意の正整数 j <= n に対して, $D_q x^n = [n]x^{n-1}$ より,

$$(D_q^j f)(x) = [n][n-1]\cdots[n-j+1]x^{n-j}$$

となるので,

$$(D_q^j f)(1) = [n][n-1]\cdots[n-j+1]$$

が得られる. □

Lemmma 2.3.3.7 ([1] p15 Example (5.5)) $n \in \mathbb{Z}_{>0}$ について,

$$(x+a)_q^n = \sum_{j=0}^{n} \begin{bmatrix} n \\ j \end{bmatrix} q^{j(j-1)/2} a^j x^{n-j}$$

が成り立つ. この式は Gauss's binomial formula と呼ばれる.

Proof. $f = (x+a)_q^n$ とすると, 任意の正整数 $j <= n$ に対して,

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

であり, また

$$(x+a)_q^m = (x+a)(x+qa)\cdots(x+q^{m-1}a)$$

から, $(0+a)_q^m = a \cdot qa \cdots q^{m-1}a = q^{m(m-1)/2}a^m$ となるので,

$$(D_q^j f)(0) = [n][n-1]\cdots[n-j+1]q^{(n-j)(n-j-1)/2}a^{n-j}$$

が成り立つ. よって, Theorem 2.3.3.2 において, $f = (x+a)_q^n$, $c = 0$ として,

$$(x+a)_q^n = \sum_{j=0}^{n} \begin{bmatrix} n \\ j \end{bmatrix} q^{(n-j)(n-j-1)/2} a^{n-j} x^j$$

が得られる. この式の右辺において j を $n-j$ に置き換えることで,

$$\begin{bmatrix} n \\ n-j \end{bmatrix} = \frac{[n]!}{[n-j]![n-(n-j)]!} = \frac{[n]!}{[j]![n-j]!} = \begin{bmatrix} n \\ j \end{bmatrix}$$

に注意すれば,

$$(x-a)_q^n = \sum_{j=0}^{n} \begin{bmatrix} n \\ j \end{bmatrix} q^{j(j-1)/2} a^j x^{n-j}$$

が成り立つ. □

この二つの等式の形式化はそれぞれ

```
Lemma q_Taylorp_pow n : (∀ n, qfact n ≠ 0) →
  'X^n = \sum_(0 ≤ i < n.+1) (qbicoef n i *: qbinom_pos_poly 1 i).

Definition qbicoef n j := qfact n / (qfact j * qfact (n - j)).
Theorem Gauss_binomial' a n : (∀ n, qfact n ≠ 0) →
  qbinom_pos_poly (-a) n =
  \sum_(0 ≤ i < n.+1) (qbicoef n i * q ^+ ((n - i) * (n - i - 1))./2 * a ^+ (n - i)) *: 'X^i.
Theorem Gauss_binomial a n : (∀ n, qfact n ≠ 0) →
  qbinom_pos_poly (-a) n =
  \sum_(0 ≤ i < n.+1) (qbicoef n i * q ^+ (i * (i - 1))./2 * a ^+ i) *: 'X^(n - i).
```

となる.

Remark 2.3.3.8 Gauss_binomial' の証明は

```
Proof.
  move⇒ Hfact.
  rewrite (q_Taylorp n (qbinom_pos_poly (-a) n) 0) //; last first.
    by rewrite qbinom_size.
  under eq_big_nat ⇒ i /andP [_ Hi].
```

```
    rewrite hoDqp'_qbinom0 //.
    rewrite [(qbinom_pos_poly 0 i / (qfact i)%:P)]mulrC.
    rewrite polyCV.
    rewrite scalerAl scale_constpoly.
    have → : qbicoef n i * qfact i * q ^+ ((n - i) * (n - i - 1))./2 *
              a ^+ (n - i) / qfact i =
              qbicoef n i * q ^+ ((n - i) * (n - i - 1))./2 * a ^+ (n - i).
      rewrite -!mulrA; f_equal; f_equal.
      rewrite mulrC -mulrA; f_equal.
      by rewrite denomK.
    rewrite mul_polyC qbinom_x0.
  over.
  done.
Qed.
```

となっており, q_Taylor ではなく q_Taylorp において c = 0 として証明している. q_Taylor には c != 0 という前提があるため, この証明には使えない.

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

[3] 萩原 学/アフェルト・レナルド, *Coq/SSReflect/Mathcomp*, 森北出版, 2018

[4] https://coq.inria.fr/distrib/current/stdlib/

[5] https://github.com/math-comp/math-comp