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

2023 年 1 月 2 日

序文

本論文の主結果は, 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 Victor Kac, Pokman Cheung の *Quantum Calculus* [1] の 4 章 (4.1) 式の q-Taylor 展開, 及びその系として得られる Gauss's binomial formula の形式化を目標としている. 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. 形式化したコード全体は `https://github.com/nakamurakaoru/q-analogue` [2] にある.

本論文の構成は, [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ある. 証明の方針等は基本的に [1] の通りであるが, 4.3 節では一部 [1] から離れ, 多項式として q-微分や q-二項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-類似は, 学部 4 年次に卒業研究のテーマとして扱ったものであり, 実数パラメータ q, 実数上の関数 f に対して

$$D_q f(x) \coloneqq \frac{f(qx) - f(x)}{(q-1)x}$$

で定義される q-微分を出発点とし, この q-微分に対してうまく振る舞い, かつ q を極限で 1 に近づけると通常の定義に一致するように数学の諸概念を一般化するものである. 例えば, x^n を定義に沿って q-微分すると,

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

となる. 通常の微分では, $(x^n)' = nx^{n-1}$ となることと比較して,

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

を自然数の q-類似と定める. 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場合があることである. 例えば, Jacobi 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 ([8] p144 (3.47) 等を参照), 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 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. Mizar, Isabelle/HOL 等他にも定理証明支援系存在するが, 修士 1 年次後期に履修した授業で Coq の使い方を学んだため, 形式化に利用した. 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. Coq は型付き λ 計算という理論に基づいている. 修士 1 年次に少人数クラスで学習した H.P.Barendregt の *Lambda Calculi with Types* [9] に沿って, 型付き λ 計算の概要について 2 で述べる. 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 3.2 節で説明するが, より詳細な情報については萩原 学/アフェルト・レナルドの *Coq/SSReflect/Mathcomp* [3] 等を参照のこと.

今後の展望としては, まずはこれまでに形式化した q-類似の各概念が $q \to 1$ としたときに通常の数学の概念に一致することの形式化を行いたい. このためには, 現在開発中のライブラリである mathcomp analysis [7] を用いる必要がある. また, このライブラリを用いると無限和に関する形式化も可能であるため, 上で紹介した Jacobi の三重積や, 無限和を用いて定義される指数関数や三角関数の q-類似の形式化にも挑戦していきたい. ただし, 無限に関わる議論の形式化は通常の数学の概念を無限に拡張するよりもさらに煩雑になることには注意が必要である.

目 次

1 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,

$$(x+a)_q^n = \sum_{j=0}^{n} \left[\begin{array}{c} n \\ j \end{array} \right] q^{\frac{j(j-1)}{2}} a^j x^{n-j}$$

が成り立つことの形式化を目標としている.

2 型付き λ 計算

2.1 λ 計算

まず型のない λ 計算を定義する. 初めに, λ 計算がどのようなものなのかについての概要を説明し, その後厳密な定義に移る.

λ 計算の概要

λ 計算には, 抽象と適用の 2 つの基本的な操作がある. まず, 抽象については, 「式から関数を作る操作」と捉えることができる. M を λ 計算における式 (λ 計算においてはこれを λ 項と呼ぶ) だとすると,

$$\lambda x.M$$

で, 「x を変数とする関数」を表すことになる. 例えば, M が $x^2 + 3xy + 4$ という式であれば,

- $\lambda x.(x^2 + 3xy + 4)$
- $\lambda xy.(x^2 + 3xy + 4)$
- $\lambda z.(x^2 + 3xy + 4)$

はそれぞれ, 1 つ目は $x \mapsto (x^2+3xy+4)$ という x についての 2 次関数, 2 つ目は $(x,y) \mapsto (x^2+3xy+4)$ という x, y についての 2 変数関数を表す. 3 つ目は, $(x^2 + 3xy + 4)$ は変数 z を含まないので, 定数関数を表すことになる.

もう 1 つの操作である適用は, 2 つの λ 項 M と N を並べて,

$$MN$$

と書かれ, 直観的には「関数 M に値 N を代入する」ことを示している. 例えば, M が $\lambda x.(3x+2)$, N が 4 であれば,

$$(\lambda x.(3x+2))\, 4 = 3 \cdot 4 + 2\ (= 14)$$

となる. 一般には, $[x := N]$ で x に N を代入することを表すとして,

$$(\lambda x.M)\, N = M[x := N]$$

と書く.

λ 計算の定義

2.2 $\lambda\!\rightarrow$ と $\lambda 2$

3 Coq

3.1 λ-cube と CIC

3.2 Coq の使い方

この節では Coq のコマンドとタクティックの使い方について述べる. ここでは, タクティックは証明の中でコンテクスト (変数や仮定) やゴール (証明すべき主張) を変形させるもの, コマンドはタクティック以外のものとして扱う. まず, よく使うコマンドについて説明する.

- Require Import
  ライブラリを読み込むためのコマンドである.
  `From mathcomp Require Import ssreflect` であれば, ライブラリ群 mathcomp から ssreflect を読み込んでいる.

- `Variable`
  `Variable [変数]: [型]` で, 特定の型を持つ変数を宣言できる.
  `Variable n: nat`
  で, 変数 `n` が自然数型 `nat` の要素であることを表している. `Section/End` コマンドと組み合わせることで, `End` まで同じ意味で扱われ, `End` 以降は効力を失う. 同時に複数の変数を宣言することもできる. その場合は
  `Variables [変数] [変数] ⋯ [変数]: [型]`
  と書く (ただし, Coq の中では Variable と Variables に違いは無い).

- `Hypothesis`
  `Hypothesis [仮定名]: [仮定]` で仮定を置くことができる. `Variable` 同様, `Section/End` と組み合わせることで, セクション内共通の仮定を置くことができる.

- `Definition`
  新たに関数を定義するためのコマンドで,
  `Definition [定義名] ([引数]: [引数の型]): [定義の型] := [定義名の定義式]`
  という形で用いる.

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

  であれば, `dq` が定義の名前, `f`, `x` が引数, `R -> R` が `f` の型であり, `f (q * x) - f x` が `dq` を定義する式である. また, `x` と `dq` そのものの型は推論できるため省略できる.

- `Lemma`
  補題を宣言するためのコマンドで,
  `Lemma [補題名] ([引数]: [引数の型]): [補題の主張]` という形である.

  ```
  Lemma Dq_pow n x : x ≠ 0 → Dq (fun x ⇒ x ^ n) x = qnat n * x ^ (n - 1).
  ```

  であれば, `Dq_pow` が補題名, `n`, `x` が引数, `:`以降が補題の主張である.
  `Lemma` の代わりに `Theorem`, `Corollary` 等でも同じ機能をもつ.

- `Proof/Qed`
  `Proof` は `Lemma` の後に書いて補題の主張と証明を分ける (実際には省略可能). 証明を完了させて `Qed` を書くことで, 他の補題の証明に使えるようになる.

次に, タクティックについて述べる. よく使われるタクティックは move, apply, rewrite の 3 つである.

- move
  move=> H でゴールの前提に H という名前をつけてコンテクストに移動する. また move: H で補題 H もしくはコンテクストに存在する H をゴールの前提に移す.

- apply
  補題 *lem* が P1 → P2 という形で, ゴールが P2 のとき, apply *lem* でゴールを P1 に変える.

- rewrite
  def が定義のとき, rewrite /*def* でゴールに出現している *def* を展開する. また, 補題 *lem* が A = B という形のとき, rewrite *lem* でゴールに出現する A を B に書き換える. 更に, rewrite *lem* in H で, コンテクストの H に出現する A を B に書き換える.

以下, 2 つの具体例を用いて Proof 内でのタクティックの使い方を説明する.

Example 3.2.1 モーダスポーネンス
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を Coq で証明する. まずこの主張を形式化すると以下の通り.

```
From mathcomp Require Import ssreflect.

Lemma modus_ponens (P Q : Prop) : (P → Q) ∧ P → Q.
```

このとき, Coq のゴールエリア (コンテクストとゴールが表示される画面) は以下の通りである.

```
1 subgoal
P, Q : Prop
--------------------------------
(P → Q) ∧ P → Q
```

---の上がコンテクスト, 下がゴールである. 1 行目の 1 subgoal はゴールが 1 つであることを示している. 用いるタクティックによってはゴールが増えることもある. 一般に命題 P_1, P_2, P_3 について $P_1 /\backslash P_2 \to P_3$ と $P_1 \to P_2 \to P_3$ は同じ意味であり, この書き換えは move=> [] で行える.

```
1 subgoal
P, Q : Prop
--------------------------------
(P → Q) → P → Q
```

ゴールに前提 (P → Q) があるため, move=> pq で pq という名前をつけてコンテクストに移動する.

```
1 subgoal
P, Q : Prop
pq : P → Q
--------------------------------
P → Q
```

まだ前提 P があるため, move=> p で p と名付けてコンテクストに移動する.

```
1 subgoal
P, Q : Prop
pq : P → Q
p : P
------------------------------
Q
```

ゴールが Q であり, コンテクストに P → Q という仮定 pq があるので, apply pq でゴールを P に書き換える.

```
1 subgoal
P, Q : Prop
pq : P → Q
p : P
------------------------------
P
```

ここまで来ると, ゴールが P であり, コンテクストに P があるため, by [] で証明を終了する.

```
No more subgoals.
```

ゴールエリアに No more subgoals. と表示されれば証明は終了であり, Qed を書くことで補題として登録されることになる.
以上をまとめると以下のようになる.

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

説明のため細かく 1 行づつ書いたが, 複数の move はまとめられること, あるタクティックによりゴールが自明なもの (コンテクストに存在する, A = A である, 計算から簡単に示されるなど) になる場合はそのタクティックの前に by をつけることで証明を終了させられることを用いれば, 次のように短くすることができる.

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

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.

Example 3.2.2 代入計算
自然数 m, n について,

$$m = 0 \Longrightarrow m + n = n$$

という簡単な代入に関する計算を証明してみる. まず主張を形式化する.

```
Lemma substitution m n : m = 0 → m + n = n.
```

このとき, ゴールエリアは以下の通りである.

```
1 subgoal
m, n : nat
------------------------------
m = 0 → m + n = n
```

まず `m = 0` という前提を `move=> Hm` で `Hm` という名前をつけてコンテクストに移動する.

```
1 subgoal
m, n : nat
Hm : m = 0
------------------------------
m + n = 0
```

`rewrite` は補題だけでなく仮定を使った書き換えも行えるため, `rewrite Hm` でゴールの `m` を `0` に書き換える.

```
1 subgoal
m, n : nat
Hm : m = 0
------------------------------
0 + n = n
```

次に, `0 + n` を `n` に書き換えたい. mathcomp の ssrnat に `add0n` という,
`forall n : nat, 0 + n = n`
に対応する補題があるため, `rewrite add0n` を実行する.

```
1 subgoal
m, n : nat
Hm : m = 0
------------------------------
n = n
```

このとき, ゴールは同じもの同士の等号であるため, 自明に成り立つ, つまり by [] で終了する.

```
No more subgoals.
```

この証明をまとめると以下の通りである.

```
Lemma substitution m n : m = 0 → m + n = n.
Proof.
  move⇒ Hm.
  rewrite Hm.
  rewrite add0n.
  by [].
Qed.
```

複数の rewrite がまとめられることと by の使い方から, より短く次のように書ける.

```
Lemma substitution m n : m = 0 → m + n = n.
Proof.
  move⇒ Hm.
  by rewrite Hm add0n.
Qed.
```

更に, 前提が A = B の形であるとき, move→ でゴールの A を B に書き換えられること, ; で異なるタクティックをつなげられることを用いれば以下のように書くこともでできる.

```
Lemma substitution m n : m = 0 → m + n = n.
Proof. by move→; by rewrite add0n. Qed.
```

Remark 3.2.3 正確には, add0n は

```
left_id 0 addn
```

という補題であり, addn は自然数同士の加法である. left_id の定義は

```
fun (S T : Type) (e : S) (op : S → T → T) ⇒ ∀ x : T, op e x = x
```

であり, 単位元を左から作用させても元のままであるということを一般的に定義している. add0n は left_id の e に 0 を, op に addn を入れたものであるので,

```
∀ x : nat, addn 0 x = x
```

となる.

このように, Coq での証明は, ゴールを自明な形になるまで繰り返し書き変えていくやり方が基本である.

4 形式化

4.1 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 4.1.1 ([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,

$$d_q f(x) \coloneqq f(qx) - f(x)$$

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

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

と定める.

この定義を形式化するためまずは実数の形式化を行う. 今回は mathcomp の ssrnum にある `rcfType` の要素を実数として用いることにする. mathcomp はある型を構成するために他の型を用いているため, ヒエラルキー (階層構造) がある. 通常の数学において体が環の性質を, 環が群の性質を引き継ぐように, mathcomp でもより一般の型の性質を引き継いでいる. `rcfType` も多くの性質を引き継いでおり, 特に `GRing` の性質を持っていることが重要である. この定義の形式化は以下の通りである.

```
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.
```

Remark 4.1.2 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 4.1.3 ([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 4.1.4 ([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 by apply mulf_neq0.
  rewrite [in x ^ n](_ : n = (n -1) +1) //; last 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, red_frac_r は,

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

という自分で用意した補題である (このように直接 q-類似に関係はしないが, 形式化をするために用意した補題は [2] に q_tool.v というファイルにまとめている). この補題の本質は $z/z = 1$ という約分計算であり, mathcomp の ssralg の補題

```
divff : ∀ (F : fieldType) (x : F), x ≠ 0 → x / x = 1
```

を用いているため z ≠ 0 という仮定が必要になる. よって, Dq_pow にももともとはなかった $x \neq 0$ という前提を加えている. 今後も補題を形式化するにあたって, その証明の中で約分を行う際には 0 でないという前提を付け加えることになる.

Remark 4.1.5 qnat という名前であるが, 実際には n の型は nat ではなく int 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. 一般に, もとの数学の証明と比べてその形式化の方が長くなる.

4.2 $(x-a)^n$ 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 4.2.1 ([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 4.2.2 $n \in \mathbb{Z}_{>0}$ 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.

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

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以下の通り.

```
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 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.
```

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 4.2.3 ([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, 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 4.2.4 ([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_pos (q ^ ((Negz n) + 1) * a) n x.
```

となる. ここで, Negz n とは Negz n = - n.+1 をみたすものであるので, (Negz n) + 1 は-n となり, [1] の定義と一致する. また 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 qbinom_pos の定義を以下のように整数に拡張する.

```
Definition qbinom a n x :=
  match n with
  | Posz n0 ⇒ qbinom_pos a n0 x
  | Negz n0 ⇒ qbinom_neg a n0.+1 x
  end.
```

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見ていく.

Proposition 4.2.5 ([1] p10 Proposition 3.2) $m, n \in \mathbb{Z}$ について, Proposition 4.2.3 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次のようになる.

```
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.
```

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 については以下の通り.

```
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.
```

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.
次に q-微分について見ていく.

Proposition 4.2.6 ([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 4.2.2 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 4.2.4 と, 商の微分公式の q-類似版である

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

及び Proposition 4.2.2 を用いて示される. □

[1] と同じ方針で証明する. まず, $n = 0$ のときは次の通り.

```
Lemma Dq_qbinomn0 a x :
  Dq (qbinom a 0) x = qnat 0 * qbinom a (- 1) x.
Proof. by rewrite Dq_const qnat0 mul0r. Qed.
```

ここで, Dq_const は

```
Lemma Dq_const x c : Dq (fun x ⇒ c) x = 0.
```

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, $n < 0$ のときは以下のようになる.

```
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 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.
```

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 4.2.4 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. これらをまとめて以下のように形式化できる.

```
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.
```

case: n で n が 0 以上か負かで場合分けを, 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_pos, Dq_qbinom_neg を使っていることが見て取れる.

4.3 関数から多項式へ

ここまでは [1] の定義に沿って関数に対して q-微分を定義し, また関数として $(x-a)_q^n$ を定義してきたが, q-Taylor 展開や Gauss's binomial formula を形式化するに当たって多項式に対する q-微分と多項式としての $(x-a)_q^n$ を改めて定義する.
ここで, Coq における多項式の扱いについて説明する.

- {poly T} で T 係数多項式を表す型となる. T は ringType でなくてはならないが, rcfType は ringType の構造を引き継いでいるため, 今回用いている R に対して {poly R} が定義できる.
- {poly R} は ring と lmodType の構造を持っている. よって, 今まで使ってきた ring に対する補題である addrC や mulrA などがそのまま使え, またスカラー倍 a *: p(ここで a: R, p: {poly R} である) も定義される.
- \poly_(i < n) E(i) で, 次数が $n-1$ 次以下, i 次の係数が $E(i)$ である多項式表す.
- c%:P で定数 c のみからなる単項式を表す.
- 'X で変数 x のみからなる単項式を表す.
- p`_i で多項式 p の i 次の係数を表す.
- size p で多項式 p の次数 +1 を表す.
- p.[x] で多項式 p の x での値を表す.

より詳細な内容については mathcomp の poly.v [6] を参照のこと.
この {poly R} を用いて Dq や qbinom を定義し直していく. まず, q-微分について, 多項式に対する d_q を以下のように定義し直す.

```
Definition scale_var (p : {poly R}):= \poly_(i < size p) (q ^ i * p`_i).
Definition dqp p := scale_var p - p.
```

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. また, dqp は多項式に対しての 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.
```

この dqp を用いて, 多項式に対する D_q を定義する.

```
Definition Dqp p := dqp p %/ dqp 'X.
```

p %/ p' は多項式 p を多項式 p' で割った商を表している. この定義だけでは dqp を dq 'X で割った余りが 0 でない可能性があるため, q-微分の正しい形式化である保証がない. しかし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. よってこのことを証明しておく.

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

ここで, 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.
```

また, dqp のときと同様, 多項式に対しての D_q と同じであることを確認しておく.

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

Remark 4.3.1 Dqp_Dqp' には特に条件がなく, Dqp'_DqE には $x \neq 0$ という条件がついている. この違いは, 前者は'X / 'X = 1%:P という多項式での約分を, 後者は x / x = 1 という実数での約分を行っているということから生じている. 後で詳しく述べるが, 約分の際に条件が必要なくなることが多項式で考える利点の一つである.

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].
```

更に, このように定義した Dqp と qbinom_pos_poly に対しても Proposition 4.2.2 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 4.3.2 証明の方針はこれまでの関数としての場合と同じだが, Dq_prod'(q-微分の積の法則)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. ここで証明の冒頭を抜き出すと以下のようになっている.

```
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 /eqP.
    rewrite size_poly_eq0.
    move/eqP →.
    by rewrite mul0r scale_varC mul0r.
  elim: n p ⇒ [|n IH] p Hsize.
  ...
Qed.
```

pose n := size p. で多項式 p の size を n と置いており, have: (size p <= n)%N by []. で size p が n 以下という自明な主張をあえて置いているが, これは多項式の size に関する帰納法を用いるためである. このように, 当たり前の内容を明示的に書かなければならないことに加え, 形式化するための証明の構造を工夫しなければならない場合もある.

多項式で考える理由は, q-Taylor 展開が多項式に対する定理であることに加え, 以下の 2 つの目的がある.

1. $x/x = 1$ の計算に $x \neq 0$ という条件が必要ない
   先に見たように, Coq で約分の計算, つまり $x/x = 1$ を行う際には $x \neq 0$ という条件が必要である. よって, 実数 x 対して $x/x = 1$ を計算する場合, 後から $x = 0$ を代入することはできない. しかし, 多項式で考える場合, `'X` は単項式であるためゼロ多項式とは異なるので, `'X ≠ 0` という条件は自動的にみたされることになり, `'X / 'X = 1%:P` の計算には特に条件が必要ない. よって, `'X` で約分した後でも 0 での値を計算できる. 例えば $D_q(x+a)_q^n = [n](x+a)_q^{n-1}$ という計算には x での約分が必要であるが, 多項式として考える場合には上の計算をした後でも 0 での値を求めることができる. この値は本論文の目的である Gauss's binomial formula の証明に必要である.

2. $q = 0$ のとき高階 D_q が定義できる
   $q = 0$ のときに 2 階 D_q を計算してみると

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

   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もともとの `dq` を関数の引数に対して各点ごとに定義しているからであり, 多項式の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.

4.4 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 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 4.4.1 ([1] p5 Theorem 2.1) $\mathbb{K} \coloneqq \mathbb{R}$ または $\mathbb{C}$, $V \coloneqq \mathbb{K}[x]$ 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.

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

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.

- `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.
  ```

  この主張には係数列 `c` の一意性は含まれていないため, 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

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

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 4.4.2 ([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 4.4.1 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 Proposition4.2.2 から分かる. □

前節で準備した `Dqp`, `qbinom_pos_poly` を用いて Theorem 4.4.2 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.
```

Remark 4.4.3 約分のための $c \neq 0$ という条件に加え, 高階 D_q を扱うため前述の通り $q \neq 0$ も必要となる. 具体的には高階 `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`(Inductive Hypothesis, 帰納の仮定) を使う際に $q * x \neq 0$ という条件が必要となる.

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 4.4.4 ([1] p12 Example (4.4)) $n \in \mathbb{Z}_{>0}$ について,

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

が成り立つ.

Proof. Theorem 4.4.2 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 4.4.5 ([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 4.4.2 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次の通り.

```
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).
```

`Gauss_binomial'` は `j` を `n - j` に置き換える前の等式である.

Remark 4.4.6 `Gauss_binomial'` は `q_Taylorp` において `c = 0` として証明している. `q_Taylorp` では約分の計算をしているが, 多項式を用いて定義しているため 0 での値を計算できる.

参考文献

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

[2] https://github.com/nakamurakaoru/q-analogue

[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

[6] https://github.com/math-comp/math-comp/blob/master/mathcomp/algebra/poly.v

[7] https://github.com/math-comp/analysis

[8] 梅村 浩, 『楕円関数論 楕円曲線の解析学』, 東京大学出版会, 2000.

[9] H.P.Barendregt, *Lambda Calculi with Types*