

The Algebras H and H^* . Let $q = e^{\hbar\epsilon\gamma}$ and set $H = \langle a, x \rangle / ([a, x] = \gamma x)$ with

$$A = e^{-\hbar\epsilon a}, \quad xA = qAx, \quad S_H(a, A, x) = (-a, A^{-1}, -A^{-1}x),$$

$$\Delta_H(a, A, x) = (a_1 + a_2, A_1A_2, x_1 + A_1x_2)$$

and dual $H^* = \langle b, y \rangle / ([b, y] = -\epsilon y)$ with

$$B = e^{-\hbar\gamma b}, \quad By = qyB, \quad S_{H^*}(b, B, y) = (-b, B^{-1}, -yB^{-1}),$$

$$\Delta_{H^*}(b, B, y) = (b_1 + b_2, B_1B_2, y_1B_2 + y_2).$$

Pairing by $(a, x)^* = (b, y) \Leftrightarrow \langle B, A \rangle = q$ making $\langle y^i b^j, a^j x^k \rangle = \delta_{ij} \delta_{ki} j! [k]_q!$ so $R = \sum \frac{y^k b^j \otimes a^i x^k}{j! [k]_q!}$.

The Algebra QU . Using the Drinfel'd double procedure, $QU_{\gamma, \epsilon} := H^{*cop} \otimes H$ with $(\phi f)(\psi g) = \langle \psi_1 S^{-1} f_3 \rangle \langle \psi_3, f_1 \rangle \langle \phi \psi_2 \rangle \langle f_2 g \rangle$ and

$$S(y, b, a, x) = (-B^{-1}y, -b, -a, -A^{-1}x),$$

$$\Delta(y, b, a, x) = (y_1 + y_2 B_1, b_1 + b_2, a_1 + a_2, x_1 + A_1 x_2).$$

Note also that $t := \epsilon a - \gamma b$ is central and can replace b , and set $QU = QU_\epsilon = QU_{1, \epsilon}$.

The 2D Lie Algebra. One may show* that if $[a, x] = \gamma x$ then $e^{\epsilon x} e^{a\alpha} = e^{a\alpha} e^{\epsilon^{-\gamma\alpha} \xi x}$. Ergo with

$$SW_{ax} \begin{array}{c} \curvearrowright \\ \curvearrowleft \end{array} S(a, x) \begin{array}{c} \xrightarrow{O_{ax}} \\ \xleftarrow{O_{xa}} \end{array} U(a, x)$$

we have $\widetilde{SW}_{ax} = e^{a\alpha + \epsilon^{-\gamma\alpha} \xi x}$.

* Indeed $xa = (a - \gamma)x$ thus $xa^n = (a - \gamma)^n x$ thus $x e^{a\alpha} = e^{\alpha(a-\gamma)x} = e^{-\gamma\alpha} e^{a\alpha} x$ thus $x^n e^{a\alpha} = e^{a\alpha} (e^{-\gamma\alpha})^n x^n$ thus $e^{\epsilon x} e^{a\alpha} = e^{a\alpha} e^{-\gamma\alpha \xi x}$.

Faddeev's Formula (In as much as we can tell, first appeared without proof in Faddeev [Fa], rediscovered and proven in Quesne [Qu], and again with easier proof, in Zagier [Za]). With $[n]_q := \frac{q^n - 1}{q - 1}$, with $[n]_q! := [1]_q [2]_q \cdots [n]_q$ and with $e_q^x := \sum_{n \geq 0} \frac{x^n}{[n]_q!}$, we have

$$\log e_q^x = \sum_{k \geq 1} \frac{(1-q)^k x^k}{k(1-q^k)} = x + \frac{(1-q)^2 x^2}{2(1-q^2)} + \dots$$

Proof. We have that $e_q^x = \frac{e^{qx} - e^x}{qx - x}$ ("the q -derivative of e_q^x is itself"), and hence $e_q^{qx} = (1 + (1-q)x)e_q^x$, and

$$\log e_q^{qx} = \log(1 + (1-q)x) + \log e_q^x.$$

Writing $\log e_q^x = \sum_{k \geq 1} a_k x^k$ and comparing powers of x , we get $q^k a_k = -(1-q)^k / k + a_k$, or $a_k = \frac{(1-q)^k}{k(1-q^k)}$. □

A Full Implementation.

Utilities

```
CF[sd_SeriesData] := MapAt[CF, sd, 3];
CF[ε_] := ExpandDenominator@ExpandNumerator@Together[
  Expand[ε] /. e^x e^y -> e^{x+y} /. e^x -> e^{CF[x]}];
```

```
Kδ /: Kδ_{i,j} := If[i == j, 1, 0];
E /: E[L1_, Q1_, P1_] ≡ E[L2_, Q2_, P2_] :=
  CF[L1 == L2] ∧ CF[Q1 == Q2] ∧ CF[Normal[P1 - P2] == 0];
E /: E[L1_, Q1_, P1_] E[L2_, Q2_, P2_] :=
  E[L1 + L2, Q1 + Q2, P1 + P2];
E[L_, Q_, P_]_{sk} := E[L, Q, Series[Normal@P, {ε, 0, sk}]]];
```

Zip and Bind

```
{t*, b*, y*, a*, x*, z*} = {τ, β, η, α, ξ, ζ};
{τ*, β*, η*, α*, ξ*, ζ*} = {t, b, y, a, x, z};
(u_{i*})* := (u*)_i;
```

```
collect[sd_SeriesData, ε_] :=
  MapAt[collect[#, ε_] &, sd, 3];
collect[ε_, ε_] := Collect[ε, ε];
Zip_{i}[P_] := P; Zip_{(ε, ε_{s...})}[P_] :=
  (collect[P // Zip_{(ε, ε_{s...})}, ε] /. f_ . ε^{d_} -> ∂_{(ε*, d)} f) /. ε* -> 0
QZip_{ε_{s...} List @ E[L_, Q_, P_] :=
  Module[{ε, z, zs, c, ys, ηs, qt, zrule, εrule},
    zs = Table[ε*, {ε, ε_{s...}}];
    c = CF[Q /. Alternatives @@ (ε_{s...} ∪ zs) -> 0];
    ys = CF@Table[∂_ε(Q /. Alternatives @@ zs -> 0), {ε, ε_{s...}}];
    ηs = CF@Table[∂_z(Q /. Alternatives @@ ε_{s...} -> 0), {z, zs}];
    qt = CF@Inverse@Table[Kδ_{z, ε*} - ∂_{z, ε} Q, {ε, ε_{s...}}, {z, zs}];
    zrule = Thread[zs -> CF[qt . (zs + ys)]];
    εrule = Thread[ε_{s...} -> ε_{s...} + ηs . qt];
    CF /@ E[L, c + ηs . qt . ys,
      Det[qt] Zip_{ε_{s...}}[P /. (zrule ∪ εrule)]];
U21 = {B_{i_}^{p_} -> e^{-p h γ b_i}, B_{i_}^{p_} -> e^{-p h γ b}, T_{i_}^{p_} -> e^{p h t_i},
  T_{i_}^{p_} -> e^{p h t}, A_{i_}^{p_} -> e^{p γ a_i}, A_{i_}^{p_} -> e^{p γ a}};
L2U = {e^{c_ . b_i + d_} -> B_{i_}^{-c/(h γ)} e^d, e^{c_ . b + d_} -> B^{-c/(h γ)} e^d,
  e^{c_ . t_i + d_} -> T_{i_}^{c/h} e^d, e^{c_ . t + d_} -> T^{c/h} e^d,
  e^{c_ . a_i + d_} -> A_{i_}^{c/γ} e^d, e^{c_ . a + d_} -> A^{c/γ} e^d,
  e^{ε_} -> e^{Expand@ε}};
LZip_{ε_{s...} List @ E[L_, Q_, P_] :=
  Module[{ε, z, zs, c, ys, ηs, lt, zrule, L1, L2, Q1, Q2},
    zs = Table[ε*, {ε, ε_{s...}}];
    c = L /. Alternatives @@ (ε_{s...} ∪ zs) -> 0;
    ys = Table[∂_ε(L /. Alternatives @@ zs -> 0), {ε, ε_{s...}}];
    ηs = Table[∂_z(L /. Alternatives @@ ε_{s...} -> 0), {z, zs}];
    lt = Inverse@Table[Kδ_{z, ε*} - ∂_{z, ε} L, {ε, ε_{s...}}, {z, zs}];
    zrule = Thread[zs -> lt . (zs + ys)];
    L2 = (L1 = c + ηs . zs /. zrule) /. Alternatives @@ zs -> 0;
    Q2 = (Q1 = Q /. U21 /. zrule) /. Alternatives @@ zs -> 0;
    CF /@ E[L2, Q2, Det[lt] e^{-L2-Q2}
      Zip_{ε_{s...}}[e^{L1+Q1} (P /. U21 /. zrule)]] // L2U];
```

```
B_{i}[L_, R_] := LR;
B_{is...}[L_{E}, R_{E}] := Module[{n}, Times[
  L /. Table[(v : b | B | t | T | a | x | y)_i -> v_{nei}, {i, {is}}],
  R /. Table[(v : β | τ | α | A | ξ | η)_i -> v_{nei}, {i, {is}}]
] // LZipJoin@Table[{β_{nei}, τ_{nei}, a_{nei}}, {i, {is}}] //
  QZipJoin@Table[{ε_{nei}, y_{nei}}, {i, {is}}];
B_{is...}[L_, R_] := B_{is}[L, R];
```

E morphisms with domain and range.

```
B_{is...} List @ E[d1 -> r1][L1_, Q1_, P1_], E[d2 -> r2][L2_, Q2_, P2_] :=
  E[(d1 ∪ Complement[d2, is]) -> (r2 ∪ Complement[r1, is]) @@
  B_{is}[E[L1, Q1, P1], E[L2, Q2, P2]];
E[d1 -> r1][L1_, Q1_, P1_] // E[d2 -> r2][L2_, Q2_, P2_] :=
  B_{r1 ∩ d2}[E[d1 -> r1][L1, Q1, P1], E[d2 -> r2][L2, Q2, P2]];
E[d1 -> r1][L1_, Q1_, P1_] ≡ E[d2 -> r2][L2_, Q2_, P2_] ^:=
  (d1 == d2) ∧ (r1 == r2) ∧ (E[L1, Q1, P1] ≡ E[L2, Q2, P2]);
E[d1 -> r1][L1_, Q1_, P1_] E[d2 -> r2][L2_, Q2_, P2_] ^:=
  E[(d1 ∪ d2) -> (r1 ∪ r2)] @@ (E[L1, Q1, P1] E[L2, Q2, P2]);
E_{d -> r}[L_, Q_, P_]_{sk} := E_{d -> r} @ E[L, Q, P]_{sk};
E_{[ε...]}[i_] := {ε}[i];
```

"Define" code

```
SetAttributes[Define, HoldAll];
Define[def_, defs_] := (Define[def]; Define[defs]);
```