

Implementation.

```
QZip_εs_list_simp @E[L_, Q_, P_] :=
Module[{ε, z, zs, c, ys, ηs, qt, zrule, Q1, Q2},
zs = Table[ε^i, {ε, εs}];
c = Q /. Alternatives @@ (εs ∪ zs) → 0;
ys = Table[∂_ε (Q /. Alternatives @@ zs → 0), {ε, εs}];
ηs = Table[∂_z (Q /. Alternatives @@ εs → 0), {z, zs}];
qt = Inverse@Table[Kδ_{z,c} - ∂_{z,c} Q, {ε, εs}, {z, zs}];
zrule = Thread[zs → qt. (zs + ys)];
Q2 = (Q1 = c + ηs.zs /. zrule) /. Alternatives @@ zs → 0;
simp /@ E[L, Q2, Det[qt] e^{-Q2} Zip_εs[e^{Q1} (P /. zrule)]];
QZip_εs_list := QZip_εs_cf;
```

```
LZip_εs_list_simp @E[L_, Q_, P_] :=
Module[{ε, z, zs, c, ys, ηs, lt, zrule, L1, L2, Q1, Q2},
zs = Table[ε^i, {ε, εs}];
c = L /. Alternatives @@ (εs ∪ zs) → 0;
ys = Table[∂_ε (L /. Alternatives @@ zs → 0), {ε, εs}];
ηs = Table[∂_z (L /. Alternatives @@ εs → 0), {z, zs}];
lt = Inverse@Table[Kδ_{z,c} - ∂_{z,c} L, {ε, εs}, {z, zs}];
zrule = Thread[zs → lt. (zs + ys)];
L2 = (L1 = c + ηs.zs /. zrule) /. Alternatives @@ zs → 0;
Q2 = (Q1 = Q /. T2t /. zrule) /. Alternatives @@ zs → 0;
simp /@
E[L2, Q2, Det[lt] e^{-L2-Q2}
Zip_εs[e^{L1+Q1} (P /. T2t /. zrule)]];
LZip_εs_list := LZip_εs_cf;
```

```
Bind({L_, R_} := L R;
Bind[{is_...}[L_E, R_E] := Module[{n},
Times[
L /. Table[{v : T | t | a | x | y}_i → v_{noi}, {i, {is}}],
R /. Table[{v : ε | α | ξ | η}_i → v_{noi}, {i, {is}}]
] // LZ1PratteneTable[{ε_{noi}, v_{noi}}, {i, {is}}] //
QZip1PratteneTable[{ε_{noi}, v_{noi}}, {i, {is}}];
B_εs_list := Bind_ε; B_is_... := Bind[{is}];
Bind[{ε_E} := ε;
Bind[L5_..., εs_list, R_] := Bind_εs[Bind[L5], R];
```

A Partial To Do List.

- Complete all “docility” arguments by identifying a “contained” docile substructure.
- Understand denominators and get rid of them.
- See if much can be gained by including P in the exponential: $\mathbb{Q}^{L+Q}P \rightsquigarrow \mathbb{Q}^{L+Q+P}$?
- Clean the program and make it efficient.
- Run it for all small knots and links, at $k = 2, 3$.
- Understand the centre and figure out how to read the output.
- Execute the Drinfel’d double procedure at \mathbb{E} -level (and thus get rid of DeclareAlgebra and all that is around it!).
- Extend to sl_3 and beyond.
- Do everything with Zip and Bind as the fundamentals, without ever referring back to (quantized) Lie algebras.

References.

[BN1] D. Bar-Natan, *Balloons and Hoops and their Universal Finite Type Invariant, BF Theory, and an Ultimate Alexander Invariant*, ωεβ/KBH, arXiv:1308.1721.

[BN2] D. Bar-Natan, *Polynomial Time Knot Polynomial*, research proposal for the 2017 Killam Fellowship, ωεβ/K17.

[BNG] D. Bar-Natan and S. Garoufalidis, *On the Melvin-Morton-Rozansky conjecture*, Invent. Math. **125** (1996) 103–133.

[BNS] D. Bar-Natan and S. Selmani, *Meta-Monoids, Meta-Bicrossed Products, and the Alexander Polynomial*, J. of Knot Theory and its Ramifications **22-10** (2013), arXiv:1302.5689.

[BV] D. Bar-Natan and R. van der Veen, *A Polynomial Time Knot Polynomial*, Proc. Amer. Math. Soc., to appear, arXiv:1708.04853.

[Fa] L. Faddeev, *Modular Double of a Quantum Group*, arXiv:math/9912078.

[GR] S. Garoufalidis and L. Rozansky, *The Loop Expansion of the Kontsevich Integral, the Null-Move, and S-Equivalence*, arXiv:math.GT/0003187.

[GST] R. E. Gompf, M. Scharlemann, and A. Thompson, *Fibered Knots and Potential Counterexamples to the Property 2R and Slice-Ribbon Conjectures*, Geom. and Top. **14** (2010) 2305–2347, arXiv:1103.1601.

- Prove a genus bound and a Seifert formula.
- Obtain “Gauss-Gassner formulas” (ωεβ/NCSU).
- Relate with Melvin-Morton-Rozansky and with Rozansky-Overbay.
- Understand the braid group representations that arise.
- Find a topological interpretation. The Garoufalidis-Rozansky “loop expansion” [GR]?
- Figure out the action of the Cartan automorphism.
- Disprove the ribbon-slice conjecture!
- Figure out the action of the Weyl group.
- Do everything at the “arrow diagram” level of finite-type invariants of (rotational) virtual tangles.
- What else can you do with the “solvable approximations”?
- And with the “Gaussian zip and bind” technology?

[MM] P. M. Melvin and H. R. Morton, *The coloured Jones function*, Commun. Math. Phys. **169** (1995) 501–520.

[Ov] A. Overbay, *Perturbative Expansion of the Colored Jones Polynomial*, University of North Carolina PhD thesis, ωεβ/Ov.

[Qu] C. Quesne, *Jackson’s q-Exponential as the Exponential of a Series*, arXiv:math-ph/0305003.

[Ro1] L. Rozansky, *A contribution of the trivial flat connection to the Jones polynomial and Witten’s invariant of 3d manifolds, I*, Comm. Math. Phys. **175-2** (1996) 275–296, arXiv:hep-th/9401061.

[Ro2] L. Rozansky, *The Universal R-Matrix, Buraou Representation and the Melvin-Morton Expansion of the Colored Jones Polynomial*, Adv. Math. **134-1** (1998) 1–31, arXiv:q-alg/9604005.

[Ro3] L. Rozansky, *A Universal U(1)-RCC Invariant of Links and Rationality Conjecture*, arXiv:math/0201139.

[Vo] H. Vo, *Alexander Invariants of Tangles via Expansions*, University of Toronto Ph.D. thesis, in preparation.

[Za] D. Zagier, *The Dilogarithm Function*, in Cartier, Moussa, Julia, and Vanhove (eds) *Frontiers in Number Theory, Physics, and Geometry II*. Springer, Berlin, Heidelberg, and ωεβ/Za.

The Complete Implementation.

ωεβ/SL2Portfolio

An even fuller implementation is at ωεβ/FullImp.

Initialization / Utilities

```
$p = 2; $k = 1; $U = QU; $E := {$k, $p};
$trim := {h^{p-} /; p > $p → 0, ε^{k-} /; k > $k → 0};
qh = e^{y e^h};
T2t = {T_{i-}^{p-} → e^{p h t}_i, T_{i-}^{p-} → e^{p h t}_i};
t2T = {e^{c- t_i + b-} → T_{i-}^c/h e^b, e^{c- t_i + b-} → T_{i-}^c/h e^b, e^{ε-} → e^{Expand[ε]}};
SetAttributes[SS, HoldAll];
SS[ε_, op_] := Collect[
Normal@Series[If[$p > 0, ε, ε /. T2t], {h, 0, $p}],
h, op];
SS[ε_] := SS[ε, Together];
Simp[ε_, op_] := Collect[ε, _CU | _QU, op];
Simp[ε_] := Simp[ε, SS[#, Expand] &];
Kδ /: Kδ_{i,j} := If[i === j, 1, 0];
c_Integer_k_Integer := c + 0[e]^{k+1};
```

```
CF[ε_] := ExpandDenominator@
ExpandNumerator@
Together[Expand[ε] /. e^x- e^y- → e^{x+y} /. e^x- → e^{CF[x]}];
Unprotect[SeriesData];
SeriesData /: CF[sd_SeriesData] := MapAt[CF, sd, 3];
SeriesData /: Expand[sd_SeriesData] :=
MapAt[Expand, sd, 3];
SeriesData /: Simplify[sd_SeriesData] :=
MapAt[Simplify, sd, 3];
SeriesData /: Together[sd_SeriesData] :=
MapAt[Together, sd, 3];
SeriesData /: Collect[sd_SeriesData, specs__] :=
MapAt[Collect[#, specs] &, sd, 3];
Protect[SeriesData];
```

DeclareAlgebra