

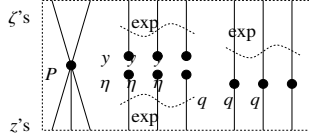
A Partial To Do List.

- Understand tr and links.
- Implement Φ, J . Determine the appropriate wt-0 ground ring.
- Implement the “dequantizers”.
- Understand denominators and get rid of them.
- Implement zipping at the log-level.
- Clean the program and make it efficient.
- Run it for all small knots and links, at $k = 3, 4$.
- Understand the centre and figure out how to read the output.
- Is the “+” really necessary in sl_{2+}^{ϵ} ? Why?
- Extend to sl_3 and beyond.
- Describe a genus bound and a Seifert formula.
- Obtain “Gauss-Gassner formulas” ($\omega\epsilon\beta/\text{NCSU}$).
- Relate with the representation theory dogma, with Melvin-Morton-Rozansky and with Rozansky-Overbay.

- Understand the braid group representations that arise.
- Relate with finite-type (Vassiliev) invariants.
- Find a topological interpretation/foundation. The Garoufalidis - Rozansky “loop expansion” [GR]?
- Figure out the action of the Cartan automorphism.
- Understand “the subspace of classical knots / tangles”.
- **Disprove the ribbon-slice conjecture!**
- Figure out the action of the Weyl group.
- Use to study “Severa quantization”.
- Do everything at the “arrow diagram” level of finite-type invariants of (rotational) virtual tangles.
- Find “internal” proofs of consistency.
- What else can you do with the “solvable approximations”?
- And with the “Gaussian compositions” technology?

Warning. Some implementation details match earlier versions of the theory.

The Zipping Theorem. If P has a finite ζ -degree and \tilde{q} is the inverse matrix of $1 - q$: $(\delta_j^i - q_j^i)\tilde{q}_k^j = \delta_k^i$, then



$$\left\langle P(z_i, \zeta^j) e^{c+\eta^i z_i + y_j \zeta^j + q_j^i z_i \zeta^j} \right\rangle = |\tilde{q}| e^{c+\eta^i \tilde{q}_i^k y_k} \left\langle P(\tilde{q}_i^k(z_k + y_k), \zeta^j + \eta^j \tilde{q}_i^j) \right\rangle.$$

The “Speedy” Engine

$\omega\epsilon\beta/\text{engine}$

Internal Utilities

Canonical Form:

```
CCF [E_] :=
  PPCF@ExpandDenominator@
  ExpandNumerator@PPTogether@Together [PPExp [
    Expand [E] /. {e^x_ e^y_ -> e^{x+y} /. e^x_ -> e^{CCF[x]}}];
CF [E_List] := CF /@ E;
CF [sd_SeriesData] := MapAt [CF, sd, 3];
CF [E_] := PPCF@Module [
  {vs = Cases [E, (y | b | t | a | x | eta | beta | tau | alpha | xi)_ , infinity] |
  {y, b, t, a, x, eta, beta, tau, alpha, xi}},
  Total [CoefficientRules [Expand [E], vs] /.
    (ps_ -> c_) -> CCF [c] (Times @@ vs^{ps})
  ];
CF [E_E] := CF /@ E;
CF [E_sp__ [E_s_____]] := CF /@ E_sp [E_s];
```

The Kronecker δ :

```
Kdelta /: Kdelta [i_, j_] := If [i === j, 1, 0];
```

Equality, multiplication, and degree-adjustment of perturbed Gaussians; $\mathbb{E}[L, Q, P]$ stands for $e^{L+Q} P$:

```
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_] $r_ := E [L, Q, Series [Normal @ P, {epsilon, 0, $k}]];
```

Zip and Bind

Variables and their duals:

```
{t*, b*, y*, a*, x*, z*} = {tau, beta, eta, alpha, xi, zeta};
{tau*, beta*, eta*, alpha*, xi*, zeta*} = {t, b, y, a, x, z};
(u_{-i}^*)^* := (u^*)_i;
```

Upper to lower and lower to Upper:

```
U21 = {B_{i-}^{p-} -> e^{-p h y b_i}, B_{p-}^{p-} -> e^{-p h y b}, T_{i-}^{p-} -> e^{p h t_i},
  T_{p-}^{p-} -> e^{p h t}, A_{i-}^{p-} -> e^{p y a_i}, A_{p-}^{p-} -> e^{p y a}};
12U = {e^{c- b_{i-} + d_{-}} -> B_{i-}^{c/(h y)} e^d, e^{c- b + d_{-}} -> B^{-c/(h y)} 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/y} e^d, e^{c- a + d_{-}} -> A^{c/y} e^d,
  e^{E-} -> e^{Expand@E}};
```

Derivatives in the presence of exponentiated variables:

```
D_b [f_] := D_b f - h y B D_b f; D_{b_i} [f_] := D_{b_i} f - h y B_i D_{b_i} f;
D_t [f_] := D_t f + h T D_t f; D_{t_i} [f_] := D_{t_i} f + h T_i D_{t_i} f;
D_alpha [f_] := D_alpha f + y A D_alpha f; D_{alpha_i} [f_] := D_{alpha_i} f + y A_i D_{alpha_i} f;
D_v [f_] := D_v f; D_{(v,0)} [f_] := f; D_{()} [f_] := f;
D_{(v,n_Integer)} [f_] := D_v [D_{(v,n-1)} [f]];
D_{(L_List, L_s_____)} [f_] := D_{(L_s)} [D_L [f]];
```

Finite Zips:

```
collect [sd_SeriesData, zeta_] :=
  MapAt [collect [# , zeta_] &, sd, 3];
collect [E_, zeta_] := PPCollect@Collect [E, zeta];
Zip_{()} [P_] := P;
Zip_{E_s} [Ps_List] := Zip_{E_s} /@ Ps;
Zip_{(E_s, E_s_____)} [P_] := PPZip [
  (collect [P // Zip_{(E_s)}, zeta_] /. f_ -> zeta^{d-} -> (D_{(zeta^{*,d})} [f])) /.
  zeta^{*} -> 0 /. ((zeta^{*} /. {b -> B, t -> T, alpha -> A}) -> 1)]
```

QZip implements the “Q-level zips” on $\mathbb{E}(L, Q, P) = e^{L+Q} P(\epsilon)$. Such zips regard the L variables as scalars.