Introducing the q-Casimir w to all orders and expressing and computing the invariant in terms of it.

Pensieve header: By Roland, June 20, 2020.

In[•]:=
```
Once[<< KnotTheory`];
```

```
Loading KnotTheory` version of September 6, 2014, 13:37:37.2841.
Read more at http://katlas.org/wiki/KnotTheory.
```

In[•]:=
```
PP_ = Identity; $k = 0; γ = 1; ħ = 1;
```

The "Speedy" Engine

Internal Utilities

Canonical Form:

In[•]:=
```
CCF[ℰ_] := ExpandDenominator@ExpandNumerator@Together[
     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_] := Module[
   {vs = Cases[ℰ, (y | b | t | a | w | x | η | β | τ | α | ω | ξ)_, ∞] ∪
      {y, b, t, a, w, x, η, β, τ, α, ω, ξ}},
   Total[CoefficientRules[Expand[ℰ], vs] /. (ps_ → c_) :> CCF[c] (Times @@ vs^ps)]
  ];
CF[ℰ_𝔼] := CF /@ ℰ; CF[𝔼_sp___[𝒮___]] := CF /@ 𝔼_sp[𝒮];
```

The Kronecker δ:

In[•]:=
```
Kδ /: Kδ_i_,j_ := If[i === j, 1, 0];
```

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

In[•]:=
```
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_]$k_ := 𝔼[L, Q, Series[Normal@P, {ϵ, 0, $k}]];
```

Zip and Bind

Variables and their duals:

```
In[•]:= {t*, b*, y*, a*, w*, x*, z*} = {τ, β, η, α, ω, ξ, ζ};
        {τ*, β*, η*, α*, ω*, ξ*, ζ*} = {t, b, y, a, w, x, z};  (u_i_)* := (u*)_i;
```

Upper to lower and lower to Upper:

```
In[•]:= U2l = {B_i_^p_. :> e^(-p ħ γ b_i), B^p_. :> e^(-p ħ γ b), T_i_^p_. :> e^(-p ħ t_i),
          T^p_. :> e^(-p ħ t), 𝒜_i_^p_. :> e^(p γ α_i), 𝒜^p_. :> e^(p γ α), Ω_i_^p_. :> e^(p ω_i), Ω^p_. :> e^(p ω)};
        l2U = {e^(c_. b_i_+d_.) :> B_i^(-c/(ħ γ)) e^d, e^(c_. b+d_.) :> B^(-c/(ħ γ)) e^d,
          e^(c_. t_i_+d_.) :> T_i^(-c/ħ) e^d, e^(c_. t+d_.) :> T^(-c/ħ) e^d,
          e^(c_. α_i_+d_.) :> 𝒜_i^(c/γ) e^d, e^(c_. α+d_.) :> 𝒜^(c/γ) e^d,
          e^(c_. ω_i_+d_.) :> Ω_i^c e^d, e^(c_. ω+d_.) :> Ω^c e^d,
          e^ℰ_ :> e^(Expand@ℰ)};
```

Derivatives in the presence of exponentiated variables:

```
In[•]:= D_b[f_] := ∂_b f - ħ γ B ∂_B f;  D_b_i_[f_] := ∂_b_i f - ħ γ B_i ∂_B_i f;
        D_t[f_] := ∂_t f - ħ T ∂_T f;  D_t_i_[f_] := ∂_t_i f - ħ T_i ∂_T_i f;
        D_α[f_] := ∂_α f + γ 𝒜 ∂_𝒜 f;  D_α_i_[f_] := ∂_α_i f + γ 𝒜_i ∂_𝒜_i f;
        D_ω[f_] := ∂_ω f + Ω ∂_Ω f;  D_ω_i_[f_] := ∂_ω_i f + Ω_i ∂_Ω_i f;
        D_v_[f_] := ∂_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,ls___}[f_] := D_{ls}[D_l[f]];
```

Finite Zips:

```
In[•]:= collect[sd_SeriesData, ζ_] := MapAt[collect[#, ζ] &, sd, 3];
        collect[ℰ_, ζ_] := Collect[ℰ, ζ];
        Zip_{}[P_] := P;
        Zip_{ζs_}[Ps_List] := Zip_ζs /@ Ps;
        Zip_{ζ_,ζs___}[P_] :=
          (collect[P // Zip_{ζs}, ζ] /. f_. ζ^d_. :> (D_{ζ*,d}[f])) /. ζ* → 0 /.
           ((ζ* /. {b → B, t → T, α → 𝒜, ω → Ω}) → 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.

$$\left\langle P(z_i,\zeta^j)e^{c+\eta^i z_i+y_j\zeta^j+q^i_j z_i\zeta^j}\right\rangle = |\tilde{q}|\left\langle P(z_i,\zeta^j)e^{c+\eta^i z_i}\Big|_{z_i\to\tilde{q}^k_i(z_k+y_k)}\right\rangle$$
$$= |\tilde{q}|e^{c+\eta^i\tilde{q}^k_i y_k}\left\langle P\left(\tilde{q}^k_i(z_k+y_k),\zeta^j+\eta^i\tilde{q}^j_i\right)\right\rangle.$$

In[ ]:=
```
QZip[ζs_List][𝔼[L_, Q_, P_]] := Module[{ζ, z, zs, c, ys, ηs, qt, zrule, ζrule, out},
  zs = Table[ζ*, {ζ, ζs}];
  c = CF[Q /. Alternatives @@ (ζs ⋃ zs) → 0];
  ys = CF@Table[∂ζ (Q /. Alternatives @@ zs → 0), {ζ, ζs}];
  ηs = CF@Table[∂z (Q /. Alternatives @@ ζs → 0), {z, zs}];
  qt = CF@Inverse@Table[Kδz,ζ* - ∂z,ζ Q, {ζ, ζs}, {z, zs}];
  zrule = Thread[zs → CF[qt.(zs + ys)]];
  ζrule = Thread[ζs → ζs + ηs.qt];
  CF /@ 𝔼[L, c + ηs.qt.ys, Det[qt] Zip[ζs][P /. (zrule ⋃ ζrule)]]];
```

LZip implements the "L-level zips" on $\mathbb{E}(L, Q, P) = \mathrm{P}e^{L+Q}$. Such zips regard all of $\mathrm{P}e^{Q}$ as a single"P". Here the z's are b and α and the ζ's are β and a.

In[ ]:=
```
LZip[ζs_List][𝔼[L_, Q_, P_]] :=
  Module[{ζ, z, zs, Zs, c, ys, ηs, lt, zrule, Zrule, ζrule, Q1, EEQ, EQ},
   (*Print["LZip"];*)
   zs = Table[ζ*, {ζ, ζs}];
   Zs = zs /. {b → B, t → T, α → 𝒜, ω → Ω};
   c = L /. Alternatives @@ (ζs ⋃ zs) → 0 /. Alternatives @@ Zs → 1;
   ys = Table[∂ζ (L /. Alternatives @@ zs → 0), {ζ, ζs}];
   ηs = Table[∂z (L /. Alternatives @@ ζs → 0), {z, zs}];
   lt = Inverse@Table[Kδz,ζ* - ∂z,ζ L, {ζ, ζs}, {z, zs}];
   zrule = Thread[zs → lt.(zs + ys)];
   Zrule = Join[zrule, zrule /.
      r_Rule :↦ ((U = r[[1]] /. {b → B, t → T, α → 𝒜, ω → Ω}) → (U /. U2l /. r //. l2U))];
   ζrule = Thread[ζs → ζs + ηs.lt];
   Q1 = Q /. (Zrule ⋃ ζrule);
   EEQ[ps___] := EEQ[ps] =
     (CF[e^-Q1 D_Thread[{zs,{ps}}][e^Q1]] /. {Alternatives @@ zs → 0, Alternatives @@ Zs → 1});
   CF@𝔼[c + ηs.lt.ys, Q1 /. {Alternatives @@ zs → 0, Alternatives @@ Zs → 1},
     Det[lt] (Zip[ζs][(EQ @@ zs) (P /. (Zrule ⋃ ζrule))] /.
         Derivative[ps___][EQ][___] :↦ EEQ[ps] /. _EQ → 1)]];
```

In[ ]:=
```
TZip[ζs_List][𝔼[L_, Q_, P_]] := Module[{ζ, z, zs, Zs, c, ys, ηs,
    lt, zrule, Zrule, ζrule, Q1, EEQ, EQ, Lnew = L, Qnew = Q, Pnew = P},
   zrule = Table[ζ* → Coefficient[L, ζ], {ζ, ζs}];
   (*Print["Tzip"];*)
   ζrule = Table[ζ → 0, {ζ, ζs}];
   Lnew = L /. U2l /. zrule /. ζrule;
   Qnew = Q /. U2l /. zrule /. ζrule; (**)
   Pnew = P /. U2l /. zrule /. ζrule;
   CF@(𝔼[Lnew, Qnew, Pnew] //. l2U)
  ];
```

```
In[•]:= B{}[L_, R_] := L R;
B{is__}[L_𝔼, R_𝔼] := Module[{n},
   Times[
      L /. Table[(v : b | B | t | T | a | w | x | y)i → vn@i, {i, {is}}],
      R /. Table[(v : β | τ | α | 𝔄 | ω | Ω | ξ | η)i → vn@i, {i, {is}}]
     ] // TZipJoin@@Table[{τn@i}, {i, {is}}] // LZipJoin@@Table[{wn@i, βn@i, an@i}, {i, {is}}] //
    QZipJoin@@Table[{ξn@i, yn@i}, {i, {is}}]];(**)
Bis__[L_, R_] := B{is}[L, R];
```

𝔼 morphisms with domain and range.

```
In[•]:= Bis_List[𝔼d1_→r1_[L1_, Q1_, P1_], 𝔼d2_→r2_[L2_, Q2_, P2_]] :=
   𝔼(d1⋃Complement[d2, is])→(r2⋃Complement[r1, is]) @@ Bis[𝔼[L1, Q1, P1], 𝔼[L2, Q2, P2]];
𝔼d1_→r1_[L1_, Q1_, P1_] // 𝔼d2_→r2_[L2_, Q2_, P2_] :=
   Br1⋂d2[𝔼d1→r1[L1, Q1, P1], 𝔼d2→r2[L2, Q2, P2]];
𝔼d1_→r1_[L1_, Q1_, P1_] ≡ 𝔼d2_→r2_[L2_, Q2_, P2_] ^:=
   (d1 == d2) ∧ (r1 == r2) ∧ (𝔼[L1, Q1, P1] ≡ 𝔼[L2, Q2, P2]);
𝔼d1_→r1_[L1_, Q1_, P1_] 𝔼d2_→r2_[L2_, Q2_, P2_] ^:=
   𝔼(d1⋃d2)→(r1⋃r2) @@ (𝔼[L1, Q1, P1] 𝔼[L2, Q2, P2]);
𝔼dr_[L_, Q_, P_]$k_ := 𝔼dr @@ 𝔼[L, Q, P]$k;
𝔼_[ℰ___][i_] := {ℰ}[[i]];
```

𝔼[Λ]

```
In[•]:= 𝔼dr_[Λ_] := CF@
   Module[{L, Λ0 = Limit[Λ, ϵ → 0]}, 𝔼dr[L = Λ0 /. (η | y | ξ | x)_ → 0, Λ0 - L, e^(Λ-Λ0)]$k /. l2U]
```

“Define” Code

Define[lhs = rhs, ...] defines the lhs to be rhs, except that rhs is computed only once for each value of $k. Fancy Mathematica not for the faint of heart. Most readers should ignore.

```
In[•]:= SetAttributes[Define, HoldAll];
Define[def_, defs__] := (Define[def]; Define[defs];);
Define[op_is__ = ℰ_] := Module[{SD, ii, jj, kk, isp, nis, nisp, sis}, Block[{i, j, k},
   ReleaseHold[Hold[
      SD[opnisp,$k_Integer, Block[{i, j, k}, opisp,$k = ℰ; opnis,$k]];
      SD[opisp, op{is},$k]; SD[opsis__, op{sis}];
     ] /. {SD → SetDelayed,
      isp → {is} /. {i → i_, j → j_, k → k_},
      nis → {is} /. {i → ii, j → jj, k → kk},
      nisp → {is} /. {i → ii_, j → jj_, k → kk_}
     }] ]]
```

Symmetric Algebra Objects

```
In[•]:= sm_{i_,j_→k_} := 𝔼_{{i,j}→{k}}[b_k (β_i + β_j) + t_k (τ_i + τ_j) + a_k (α_i + α_j) + y_k (η_i + η_j) + x_k (ξ_i + ξ_j)];
        sΔ_{i_→j_,k_} := 𝔼_{{i}→{j,k}}[β_i (b_j + b_k) + τ_i (t_j + t_k) + α_i (a_j + a_k) + η_i (y_j + y_k) + ξ_i (x_j + x_k)];
        sS_{i_} := 𝔼_{{i}→{i}}[-β_i b_i - τ_i t_i - α_i a_i - η_i y_i - ξ_i x_i];
        sϵ_{i_} := 𝔼_{{}→{i}}[0];
        sη_{i_} := 𝔼_{{i}→{}}[0];
```

```
In[•]:= sσ_{i_→j_} := 𝔼_{{i}→{j}}[β_i b_j + τ_i t_j + α_i a_j + η_i y_j + ξ_i x_j];
        sϒ_{i_→j_,k_,l_,m_} := 𝔼_{{i}→{j,k,l,m}}[β_i b_k + τ_i t_k + α_i a_l + η_i y_j + ξ_i x_m];
```

Booting Up QU

```
In[•]:= Define[aσ_{i→j} = 𝔼_{{i}→{j}}[a_j α_i + x_j ξ_i], bσ_{i→j} = 𝔼_{{i}→{j}}[b_j β_i + y_j η_i]]
```

```
In[•]:= Define[am_{i,j→k} = 𝔼_{{i,j}→{k}}[(α_i + α_j) a_k + (ℛ_j^{-1} ξ_i + ξ_j) x_k],
          bm_{i,j→k} = 𝔼_{{i,j}→{k}}[(β_i + β_j) b_k + (η_i + e^{-ϵ β_i} η_j) y_k]]
```

Three types of inverses appear below!

$\overline{R}$ is the inverse of R in the algebra $\mathbb{B}\otimes\mathbb{A}$.

P is the inverse of R as a quadratic form, like how an element of $V^*\otimes V^*$ can be the inverse of an element of $V\otimes V$.

$\overline{aS}$ is the inverse of aS as an operator form, like how an element of $V^*\otimes V$ can be the inverse of another element of $V^*\otimes V$.

```
In[•]:= Define[R_{i,j} = 𝔼_{{}→{i,j}}[ħ a_j b_i + Sum_{k=1}^{$k+1} (1 - e^{γ ϵ ħ})^k (ħ y_i x_j)^k / (k (1 - e^{k γ ϵ ħ}))],
          R̄_{i,j} = CF@𝔼_{{}→{i,j}}[-ħ a_j b_i, -ħ x_j y_i / B_i, 1 + If[$k == 0, 0, (R̄_{{i,j},$k-1})$k[3] -
              (((R̄_{{i,j},0})$k R_{1,2} (R̄_{{3,4},$k-1})$k) // (bm_{i,1→i} am_{j,2→j}) // (bm_{i,3→i} am_{j,4→j}))[3]]],
          P_{i,j} = 𝔼_{{i,j}→{}}[β_i α_j / ħ, η_i ξ_j / ħ, 1 + If[$k == 0, 0, (P_{{i,j},$k-1})$k[3] -
              (R_{1,2} // ((P_{{1,j},0})$k (P_{{i,2},$k-1})$k))[3]]]]
```

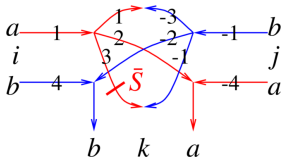
```
In[•]:= Define[aS_i = (aσ_{i→2} R̄_{1,i}) // P_{1,2},
          āS_i = 𝔼_{{i}→{i}}[-a_i α_i, -x_i ℛ_i ξ_i, 1 + If[$k == 0, 0, (āS_{{i},$k-1})$k[3] -
              ((āS_{{i},0})$k // aS_i // (āS_{{i},$k-1})$k)[3]]]]
```

(was $aS_j = \overline{R}_{i,j} \sim B_i \sim P_{i,j}$).

```
In[•]:= Define[bS_i = bσ_{i→1} R_{i,2} // aS_2 // P_{1,2},
          b̄S_i = bσ_{i→1} R_{i,2} // āS_2 // P_{1,2},
          aΔ_{i→j,k} = (R_{1,j} R_{2,k}) // bm_{1,2→3} // P_{3,i},
          bΔ_{i→j,k} = (R_{j,1} R_{k,2}) // am_{1,2→3} // P_{i,3}]
```

(was $bS_i = R_{i,1} \sim B_1 \sim aS_1 \sim B_1 \sim P_{i,1}$, $\overline{bS}_i = R_{i,1} \sim B_1 \sim \overline{aS}_1 \sim B_1 \sim P_{i,1}$).

The Drinfel'd double:



In[•]:=
```
Define[
 dm_{i,j→k} = ((sϒ_{i→4,4,1,1} // aΔ_{1→1,2} // aΔ_{2→2,3} // aS̄_3) (sϒ_{j→-1,-1,-4,-4} // bΔ_{-1→-1,-2} // bΔ_{-2→-2,-3})) //
   (P_{-1,3} P_{-3,1} am_{2,-4→k} bm_{4,-2→k})]
```

In[•]:=
```
Define[dσ_{i→j} = aσ_{i→j} bσ_{i→j},
 dϵ_i = sϵ_i,  dη_i = sη_i,
 dS_i = sϒ_{i→1,1,2,2} // (bS̄_1 aS_2) // dm_{2,1→i},
 dS̄_i = sϒ_{i→1,1,2,2} // (bS_1 aS̄_2) // dm_{2,1→i},
 dΔ_{i→j,k} = (bΔ_{i→3,1} aΔ_{i→2,4}) // (dm_{3,4→k} dm_{1,2→j})]
```

In[•]:=
```
Define[C_i = 𝔼_{{}→{i}}[0, 0, B_i^{1/2} e^{-ħ ϵ a_i/2}]_{$k},
 C̄_i = 𝔼_{{}→{i}}[0, 0, B_i^{-1/2} e^{ħ ϵ a_i/2}]_{$k},
 Kink_i = (R_{1,3} C̄_2) // dm_{1,2→1} // dm_{1,3→i},
 K̄ink_i = (R̄_{1,3} C_2) // dm_{1,2→1} // dm_{1,3→i}]
```

Note. $t = -\epsilon a + \gamma b$ and $b = t/\gamma + \epsilon a/\gamma$

In[•]:=
```
Define[b2t_i = 𝔼_{{i}→{i}}[α_i a_i + β_i (ϵ a_i + t_i) / ϒ + ξ_i x_i + η_i y_i],
 t2b_i = 𝔼_{{i}→{i}}[α_i a_i + τ_i (-ϵ a_i + ϒ b_i) + ξ_i x_i + η_i y_i]]
```

The t-Tensors

In[•]:=
```
Define[tR_{i,j} = R_{i,j} // (b2t_i b2t_j),
 tR̄_{i,j} = R̄_{i,j} // (b2t_i b2t_j),
 tm_{i,j→k} = ((t2b_i t2b_j) // dm_{i,j→k} // b2t_k),
 tC_i = (C_i // b2t_i),
 tC̄_i = (C̄_i // b2t_i),
 tKink_i = Kink_i // b2t_i,
 tK̄ink_i = K̄ink_i // b2t_i,
 tΔ_{i→j,k} = t2b_i // dΔ_{i->j,k} // (b2t_j b2t_k),
 tS_i = t2b_i // dS_i // b2t_i]
```

The w-Tensors

Use the central variable $w = \frac{1}{2} + a + \frac{x\,y}{1-T}$

```
In[•]:= a2w_i_ := 𝔼_{i}→{i}[τ_i t_i + α_i (-1/2 + w_i), (e^(-α_i) - 1) y_i x_i/(1 - T_i) + ξ_i x_i + η_i y_i, 1]
        w2a_i_ := 𝔼_{i}→{i}[τ_i t_i + (a_i + 1/2) ω_i, (1 - e^(-ω_i))/(1 - T_i) y_i x_i + ξ_i x_i + η_i y_i, 1]

In[•]:= 𝔼_{}→{i,j}[0, 0, x_i y_j - x_j y_i] // wm_{i,j→k}
Out[•]= 𝔼_{}→{k}[0, 0, 1]
```

Up to some notational annoyance the kink is exp(tw)

```
In[•]:= tKink_i // a2w_i
        tKink‾_i // a2w_i
Out[•]= 𝔼_{}→{i}[-t_i/2 + t_i w_i, 0, 1/√T_i + O[∈]^1]
Out[•]= 𝔼_{}→{i}[t_i/2 - t_i w_i, 0, √T_i + O[∈]^1]
```

The R-matrix becomes complicated! I also rescaling x by xnew = $(1-T)^{-1}$xold will help

```
In[•]:= tR_{i,j} // a2w_i // a2w_j
        tR‾_{i,j} // a2w_i // a2w_j
        w2a_i // w2a_j // tm_{i,j→k} // a2w_k
        tC_i // a2w_i
        tC‾_i // a2w_i
Out[•]= 𝔼_{}→{i,j}[-t_i/2 + t_i w_j, x_j y_i + (1 - T_i) x_j y_j/(-1 + T_j), 1 + O[∈]^1]
Out[•]= 𝔼_{}→{i,j}[t_i/2 - t_i w_j, -x_j y_i/T_i + (-1 + T_i) x_j y_j/(-T_i + T_i T_j), 1 + O[∈]^1]
Out[•]= 𝔼_{i,j}→{k}[t_k τ_i + t_k τ_j + w_k ω_i + w_k ω_j, y_k η_i + y_k η_j + x_k ξ_i + (1 - T_k) η_j ξ_i + x_k ξ_j, 1 + O[∈]^1]
Out[•]= 𝔼_{}→{i}[0, 0, √T_i + O[∈]^1]
Out[•]= 𝔼_{}→{i}[0, 0, 1/√T_i + O[∈]^1]
```

So let's define the newly found building blocks independently: (recall T = exp(-t) so the annoying $-\frac{t}{2}$ in the L part is really Sqrt[T] in the P part.)
(not quite sure I did the rescaling correctly (why would it not be T_j?))

```
Define[
 wR_{i,j} = 𝔼_{}→{i,j}[t_i w_j, (1 - T_i) x_j y_i - (1 - T_i) x_j y_j, Sqrt[T_i]],
 wR̄_{i,j} = 𝔼_{}→{i,j}[-t_i w_j, (-(1 - T_i) x_j y_i + (1 - T_i) y_j x_j)/T_i, 1/Sqrt[T_i]],
 wC_i = 𝔼_{}→{i}[0, 0, Sqrt[T_i]],
 wC̄_i = 𝔼_{}→{i}[0, 0, 1/Sqrt[T_i]],
 wm_{i,j→k} = 𝔼_{i,j}→{k}[t_k τ_i + t_k τ_j + w_k ω_i + w_k ω_j, y_k η_i + y_k η_j + x_k ξ_i + η_j ξ_i + x_k ξ_j, 1 + O[ϵ]^1]
]
```

Almost matching Γ calculus

Checking Reidemeister 1: (it is satisfied up to an overall factor of $e^{+-w\,t}$)

```
In[•]:= wR_{1,2} wC̄_3 // wm_{1,3→1} // wm_{1,2→1}
        wR̄_{1,2} wC_3 // wm_{1,3→1} // wm_{1,2→1}
        wR̄_{1,2} wC̄_3 // wm_{2,3→2} // wm_{2,1→1}
        wR_{1,2} wC_3 // wm_{2,3→2} // wm_{2,1→1}
Out[•]= 𝔼_{}→{1}[t_1 w_1, 0, 1 + O[ϵ]^1]
Out[•]= 𝔼_{}→{1}[-t_1 w_1, 0, 1 + O[ϵ]^1]
Out[•]= 𝔼_{}→{1}[-t_1 w_1, 0, 1 + O[ϵ]^1]
Out[•]= 𝔼_{}→{1}[t_1 w_1, 0, 1 + O[ϵ]^1]
```

Checking Reidemeister 2:

```
In[•]:= wR_{1,2} wR̄_{3,4} // wm_{1,3->1} // wm_{2,4→2}
Out[•]= 𝔼_{}→{1,2}[0, 0, 1 + O[ϵ]^1]
```

Checking Reidemeister 3:

```
In[•]:= (wR_{1,2} wR_{4,3} wR_{5,6} // wm_{1,4→1} // wm_{2,5→2} // wm_{3,6→3}) ≡
          (wR_{2,3} wR_{1,6} wR_{4,5} // wm_{1,4→1} // wm_{2,5→2} // wm_{3,6→3})
Out[•]= True
```

Trefoil knot

```
In[•]:= (wR_{5,1} wR_{2,6} wR_{7,3} wC_4 // wm_{1,2→1} // wm_{1,3→1} // wm_{1,4→1} // wm_{1,5→1} // wm_{1,6→1} // wm_{1,7→1})
Out[•]= 𝔼_{}→{1}[3 t_1 w_1, 0, T_1/(1 - T_1 + T_1^2) + O[ϵ]^1]
```

Let's look at the product in Γ calculus style. Caution: variables γ and ϵ are in use, use g and e instead. Taking the opposite product gives Γ calc. Provided the matrix A of Γ calculus is written as A=I+Q, where Q is the quadratic
actually used in Gaussian calculus.
First let's check out the crossings $wR_{1,2}$ and $\overline{wR}_{1,2}$ turn into the Γ calc values for the crossings. Except for

the annoying? Sqrt[T] factor.

but that's ok.

```
In[•]:= Table[Coefficient[wR1,2[[2]], yi xj], {i, {1, 2}}, {j, {1, 2}}] + IdentityMatrix[2] //
          FullSimplify // MatrixForm
        Table[Coefficient[wR1,2[[2]], yi xj], {i, {1, 2}}, {j, {1, 2}}] + IdentityMatrix[2] //
          FullSimplify // Expand // MatrixForm
```

Out[•]//MatrixForm=

$$\begin{pmatrix} 1 & 1-T_1 \\ 0 & T_1 \end{pmatrix}$$

Out[•]//MatrixForm=

$$\begin{pmatrix} 1 & 1-\frac{1}{T_1} \\ 0 & \frac{1}{T_1} \end{pmatrix}$$

```
In[•]:= (*We start with the matrix *)A = {{α, β, Θ}, {g, δ, ϵ}, {ϕ, ψ, Ξ}};(*and scalar Omega*)

In[•]:= (*Now form the relevant Q =A-I*)
        Q = {ya, yb, yS}.(A - IdentityMatrix[3]).{xa, xb, xS} // Expand
```

Out[•]= $-x_a y_a + \alpha x_a y_a + \beta x_b y_a + \Theta x_S y_a + g x_a y_b - x_b y_b + \delta x_b y_b + \epsilon x_S y_b + \phi x_a y_S + \psi x_b y_S - x_S y_S + \Xi x_S y_S$

```
In[•]:= (*Compute the product in the Gaussian way but OPPOSITE*)
        ProdResult = 𝔼{}→{a,b,S}[0, Q, Omega] // wmb,a→c
```

Out[•]= $\mathbb{E}_{\{\}\to\{c,S\}}\left[0, \frac{1}{-1+\beta}\right.$

$(x_c y_c - g x_c y_c - \beta x_c y_c + g \beta x_c y_c - \alpha \delta x_c y_c - \epsilon x_S y_c + \beta \epsilon x_S y_c - \delta \Theta x_S y_c - \phi x_c y_S + \beta \phi x_c y_S -$

$\left.\alpha \psi x_c y_S + x_S y_S - \beta x_S y_S - \Xi x_S y_S + \beta \Xi x_S y_S - \Theta \psi x_S y_S), -\frac{\text{Omega}}{-1+\beta} + O[\epsilon]^1\right]$

```
In[•]:= (*Extract the resulting newQ*)
        NewQ = Table[Coefficient[ProdResult[[2]], yi xj], {i, {c, S}}, {j, {c, S}}];
        NewQ // MatrixForm;
        (*Form the newA = NewQ+I*)
        NewA = NewQ + IdentityMatrix[2] // FullSimplify;
        NewA // MatrixForm (*et voila, the golden standard comes out.*)
```

Out[•]//MatrixForm=

$$\begin{pmatrix} g - \frac{\alpha\delta}{-1+\beta} & \epsilon - \frac{\delta\Theta}{-1+\beta} \\ \phi - \frac{\alpha\psi}{-1+\beta} & \Xi - \frac{\Theta\psi}{-1+\beta} \end{pmatrix}$$