Pensieve header: The "speedy" engine.

The "Speedy" Engine

Program

Internal Utilities

Program

Canonical Form:

Program

In[•]:=
```
CCF[ℰ_] := PPCCF@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 | η | β | τ | α | ξ)_, ∞] ⋃ {y, b, t, a, x, η, β, τ, α, ξ}},
    Total[CoefficientRules[Expand[ℰ], vs] /. (ps_ → c_) :> CCF[c] (Times @@ vs^ps)]
   ];
CF[ℰ_𝔼] := CF /@ ℰ; CF[𝔼sp___[ℰs___]] := CF /@ 𝔼sp[ℰs];
```

Program

The Kronecker δ:

Program

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

Program

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

Program

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}]];
```

Program

In[•]:=
```
𝔼3@𝔼[ω_, L_, Q_, Ps_] := CF /@ 𝔼[L, ω^-1 Q, ω^-1 (ω^-4 ϵ)^(-1+Range@Length@Ps .Ps)]$k;
𝔼4@𝔼[L_, Q_, P_] := Module[
    {ω = Normal[P]^-1 /. ϵ → 0, Ps = CoefficientList[P, ϵ]},
    CF /@ 𝔼[ω, L, ω Q, ω^(-3+4 Range@Length@Ps Ps)]];
𝔼3@𝔼sp___[as___] := 𝔼3@𝔼[as] /. 𝔼 → 𝔼sp;
𝔼4@𝔼sp___[as___] := 𝔼4@𝔼[as] /. 𝔼 → 𝔼sp;
```

Program

Zip and Bind

Program

Variables and their duals:

Program

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

Program

Finite Zips:

Program

In[•]:=
```
collect[sd_SeriesData, ξ_] := MapAt[collect[#, ξ] &, sd, 3];
collect[ℰ_, ξ_] := PP_Collect@Collect[ℰ, ξ];
Zip_{}[P_] := P;
Zip_{ξs_}[Ps_List] := Zip_ξs /@ Ps;
Zip_{ξ_,ξs___}[P_] := PP_Zip[
  (collect[P // Zip_{ξs}, ξ] /. f_. ξ^d_. :> ∂_{ξ*,d} f) /. ξ* → 0]
```

Program

QZip implements the "Q-level zips" on $\mathbb{E}(L, Q, P) = e^{L+Q} P(\epsilon)$ and/or on $\mathbb{E}(\omega, L, Q, P) = \omega^{-1} e^{L+\omega^{-1} Q} P(\omega^{-4} \epsilon)$. Such zips regard the L variables as scalars.

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

Program

In[•]:=
```
$QZipFail = False;
QZip_ξs_List@𝔼[L_, Q_, P_] := PP_QZip@Module[{ξ, z, zs, c, ys, ηs, qt, zrule, ξrule, out},
    zs = Table[ξ*, {ξ, ξs}];
    c = CF[Q /. Alternatives @@ (ξs ⋃ zs) → 0];
    ys = CF@Table[∂_ξ (Q /. Alternatives @@ zs → 0), {ξ, ξs}];
    ηs = CF@Table[∂_z (Q /. Alternatives @@ ξs → 0), {z, zs}];
    qt = CF@Inverse@Table[Kδ_{z,ξ*} - ∂_{z,ξ} Q, {ξ, ξs}, {z, zs}];
    zrule = Thread[zs → CF[qt.(zs + ys)]];
    ξrule = Thread[ξs → ξs + ηs.qt];
    out = CF /@ 𝔼[L, c + ηs.qt.ys, Det[qt] Zip_ξs[P /. (zrule ⋃ ξrule)]];
    If[¬ ($QZipFail ∨ TrueQ[out ≡ 𝔼3@QZip_ξs@𝔼4@𝔼[L, Q, P]]),
     $QZipFail = True; Print["QZip4 fail at {L,Q,P}=", {L, Q, P}]
    ];
    out
   ];
```

Program

In[ ]:=
```
$QZipFail = False;
QZip_ζs_List@𝔼[ω_, L_, Q_, Ps_] := PP_QZip4@Module[{ζ, z, zs, c, ys, ηs, qt, zrule, ζrule},
    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[Kδ_{z,ζ*} - ∂_{z,ζ}Q, {ζ, ζs}, {z, zs}];
    zrule = Thread[zs → CF[qt.(zs + ys)]];
    ζrule = Thread[ζs → ζs + ηs.qt];
    CF /@ 𝔼[ω Det[qt / ω], L, c + ηs.qt.ys, Zip_ζs[Ps /. (zrule ⋃ ζrule)]]];
```

Program

Upper to lower and lower to Upper:

Program

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}};
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^{ℰ_} :> e^{Expand@ℰ}};
```

Program

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

Program

In[ ]:=
```
LZip_ζs_List@𝔼[L_, Q_, P_] :=
  PP_LZip@Module[{ζ, z, zs, c, ys, ηs, lt, zrule, Zrule, ζrule, Q1, EEQ, EQ},
    zs = Table[ζ*, {ζ, ζs}];
    c = L /. Alternatives@@(ζs ⋃ zs) → 0;
    ys = Table[∂_ζ (L /. Alternatives@@zs → 0), {ζ, ζs}];
    ηs = Table[∂_z (L /. Alternatives@@ζs → 0), {z, zs}];
    lt = Inverse@Table[Kδ_{z,ζ*} - ∂_{z,ζ}L, {ζ, ζs}, {z, zs}];
    zrule = Thread[zs → lt.(zs + ys)];
    Zrule = zrule /. r_Rule :>
        ((U = r[[1]] /. {b → B, t → T, α → 𝒜}) → (U /. U2l /. r //. l2U)); (* not used *)
    ζrule = Thread[ζs → ζs + ηs.lt];
    Q1 = Q /. U2l /. (zrule ⋃ ζrule);
    EEQ[ps___] := EEQ[ps] = PP_"EEQ"@(CF[e^{-Q1} D[e^{Q1}, Sequence@@Thread[{zs, {ps}}]]] /.
        Alternatives@@zs → 0 //. l2U);
    CF /@ ((*CF/@*)𝔼[
        c + ηs.lt.ys, Q1 /. Alternatives@@zs → 0,
        Det[lt] (Zip_ζs[(EQ@@zs) (P /. U2l /. (zrule ⋃ ζrule))] /.
            Derivative[ps___][EQ][___] :> EEQ[ps] /. _EQ → 1)
    ] //. l2U)
  ];
```

Program

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

Program

𝔼 morphisms with domain and range.

Program

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]);
𝔼d_→r_[L_, Q_, P_]$k_ := 𝔼d→r @@ 𝔼[L, Q, P]$k;
𝔼_[ℰ___][i_] := {ℰ}[[i]];
```