Pensieve header: The "Speedy" engine, before E3E4 de-clutter.

Program

The "Speedy" Engine

Program

Internal Utilities

Program

Canonical Form:

Program

```
In[•]:= CCF[ℰ_] := PP_CCF@ExpandDenominator@ExpandNumerator@PP_Together@Together[PP_Exp[
          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_] := PP_CF@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[𝔼_sp___[ℰs___]] := CF /@ 𝔼_sp[ℰs];
```

Program

The Kronecker δ:

Program

```
In[•]:= Kδ /: Kδ_i_,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@𝔼_sp___[as___] := 𝔼3@𝔼[as] /. 𝔼 → 𝔼_sp;
        𝔼4@𝔼_sp___[as___] := 𝔼4@𝔼[as] /. 𝔼 → 𝔼_sp;
```

Program

Zip and Bind

Program

Variables and their duals:

Program

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

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

```
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_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]];
```

Program

Finite Zips:

Program

```
In[•]:= collect[sd_SeriesData, ζ_] := MapAt[collect[#, ζ] &, sd, 3];
        collect[ℰ_, ζ_] := PP_Collect@Collect[ℰ, ζ];
        Zip_{}[P_] := P;
        Zip_ζs_[Ps_List] := Zip_ζs /@ Ps;
        Zip_{ζ_,ζs___}[P_] := PP_Zip[
          (collect[P // Zip_{ζs}, ζ] /. f_. ζ^d_. :> (D_{ζ*,d}[f])) /. ζ* → 0 /.
           ((ζ* /. {b → B, t → T, α → 𝒜}) → 1)]
```

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[ ]:=
```
$QZipFail = False;
QZip_{ζs_List}@𝔼[L_, Q_, P_] := PP_QZip@Module[{ζ, z, zs, c, ys, ηs, qt, zrule, ζrule, out},
    zs = Table[ζ*, {ζ, ζs}];
    c = CF[Q /. Alternatives@@(ζs ⋃ zs) → 0];
    ys = CF@Table[∂_ζ (Q /. Alternatives@@zs → 0), {ζ, ζs}];
    ηs = CF@Table[∂_z (Q /. Alternatives@@ζs → 0), {z, zs}];
    qt = CF@Inverse@Table[Kδ_{z,ζ*} - ∂_{z,ζ}Q, {ζ, ζs}, {z, zs}];
    zrule = Thread[zs → CF[qt.(zs + ys)]];
    ζrule = Thread[ζs → ζs + ηs.qt];
    out = CF /@ 𝔼[L, c + ηs.qt.ys, Det[qt] Zip_ζs[P /. (zrule ⋃ ζrule)]];
    If[¬ ($QZipFail ∨ TrueQ[out ≡ 𝔼3@QZip_ζs@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^{p_}_{i_}· → ℯ^{-p ħ γ b_i}, B^{p_}· → ℯ^{-p ħ γ b}, T^{p_}_{i_}· → ℯ^{p ħ t_i}, T^{p_}· → ℯ^{p ħ t}, 𝒜^{p_}_{i_}· → ℯ^{p γ α_i}, 𝒜^{p_}· → ℯ^{p γ α}};
l2U = {ℯ^{c_.b_i+d_.} :↦ B_i^{-c/(ħγ)} ℯ^d, ℯ^{c_.b+d_.} :↦ B^{-c/(ħγ)} 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 $\mathbb{E}(L, Q, P) = P\mathrm{e}^{L+Q}$. Such zips regard all of $P\mathrm{e}^{Q}$ as a single"P". Here the z's are b and α and the ζ's are β and a.

Program

```
LZip_ξs_List@𝔼[L_, Q_, P_] :=
  PP_LZip@Module[{ξ, z, zs, Zs, c, ys, ηs, lt, zrule, Zrule, ξrule, Q1, EEQ, EQ},
    zs = Table[ξ*, {ξ, ξs}];
    Zs = zs /. {b → B, t → T, α → 𝒜};
    c = L /. Alternatives@@(ξs ⋃ zs) → 0;
    ys = Table[∂_ξ (L /. Alternatives@@zs → 0), {ξ, ξs}];
    ηs = Table[∂_z (L /. Alternatives@@ξs → 0), {z, zs}];
    lt = Inverse@Table[Kδ_{z,ξ*} - ∂_{z,ξ} L, {ξ, ξs}, {z, zs}];
    zrule = Thread[zs → lt.(zs + ys)];
    Zrule = zrule ~Join~
      (zrule /. r_Rule :↦ ((U = r[[1]] /. {b → B, t → T, α → 𝒜}) → (U /. U2l /. r //. l2U)));
    ξrule = Thread[ξs → ξs + ηs.lt];
    Q1 = Q /. (Zrule ⋃ ξrule);
    EEQ[ps___] := EEQ[ps] = (PP_"EEQ"@(CF[ⅇ^-Q1 D_Thread[{zs,{ps}}][ⅇ^Q1]] /.
        {Alternatives@@zs → 0, Alternatives@@Zs → 1}));
    CF /@ ((*CF/@*)𝔼[
       c + ηs.lt.ys, Q1 /. {Alternatives@@zs → 0, Alternatives@@Zs → 1},
       Det[lt] (Zip_ξs[(EQ@@zs) (P /. (Zrule ⋃ ξrule))] /.
           Derivative[ps___][EQ][___] :↦ EEQ[ps] /. _EQ → 1)
     ])
   ];
```

Program

In[•]:=

```
B_{}[L_, R_] := L R;
B_{is__}[L__𝔼, R_𝔼] := PP_B@Module[{n},
    Times[
       L /. Table[(v : b | B | t | T | a | x | y)_i → v_{n@i}, {i, {is}}],
       R /. Table[(v : β | τ | α | 𝒜 | ξ | η)_i → v_{n@i}, {i, {is}}]
      ] // LZip_Join@@Table[{β_{n@i}, τ_{n@i}, a_{n@i}}, {i, {is}}] // QZip_Join@@Table[{ℰ_{n@i}, y_{n@i}}, {i, {is}}] ];
B_is___[L_, R_] := B_{is}[L, R];
```

Program

𝔼 morphisms with domain and range.

Program

In[•]:=

```
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_]$k_ := 𝔼_d→r @@ 𝔼[L, Q, P]$k;
𝔼_[ℰ___][i_] := {ℰ}[[i]];
```