Pensieve header: Experiments with lazy evaluation.

Program

The "Lazy Evaluation" 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; $\mathbb{E}[V, L, Q, P]$ stands for $\langle e^{L+Q} P \rangle_V$:

Program

```
𝔼[L : Except[_List | _Pattern], ℰs___] := 𝔼[{}, L, ℰs];
𝔼 /: 𝔼[{}, L1_, Q1_, P1_] ≡ 𝔼[{}, L2_, Q2_, P2_] :=
  CF[L1 == L2] ∧ CF[Q1 == Q2] ∧ CF[Normal[P1 - P2] == 0];
𝔼 /: E1_𝔼 ≡ E2_𝔼 := Zip@E1 ≡ Zip@E2;
𝔼 /: 𝔼[V1_, L1_, Q1_, P1_] 𝔼[V2_, L2_, Q2_, P2_] := 𝔼[V1 ⋃ V2, L1 + L2, Q1 + Q2, P1 * P2];
𝔼 /: 𝔼[V_, L_, Q_, P_]$k_ := 𝔼[V, L, Q, Series[Normal@P, {ϵ, 0, $k}]];
```

Program

```
𝔼3@𝔼[V_, ω_, L_, Q_, Ps_] := CF /@ 𝔼[V, L, ω^-1 Q, ω^-1 (ω^-4 ϵ)^(-1+Range@Length@Ps . Ps)]$k;
𝔼4@𝔼[V_, L_, Q_, P_] := Module[
    {ω = Normal[P]^-1 /. ϵ → 0, Ps = CoefficientList[P, ϵ]},
    CF /@ 𝔼[V, 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*, ξ*, ζ*} = {t, b, y, a, x, z}; (u_i_)* := (u*)ᵢ;
```

Program

```
In[ ]:= U2l = {B_i_^p_. → 𝕖^(-p ħ γ bᵢ), B^p_. → 𝕖^(-p ħ γ b), T_i_^p_. → 𝕖^(p ħ tᵢ), T^p_. → 𝕖^(p ħ t), 𝒜_i_^p_. → 𝕖^(p γ αᵢ), 𝒜^p_. → 𝕖^(p γ α)};
        I2U = {𝕖^(c_. bᵢ_+d_.) :> Bᵢ^(-c/(ħ γ)) 𝕖^d, 𝕖^(c_. b+d_.) :> B^(-c/(ħ γ)) 𝕖^d,
            𝕖^(c_. tᵢ_+d_.) :> Tᵢ^(c/ħ) 𝕖^d, 𝕖^(c_. t+d_.) :> T^(c/ħ) 𝕖^d,
            𝕖^(c_. αᵢ_+d_.) :> 𝒜ᵢ^(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_bᵢ_[f_] := ∂_bᵢ f - ħ γ Bᵢ ∂_Bᵢ f;
        D_t[f_] := ∂_t f + ħ T ∂_T f; D_tᵢ_[f_] := ∂_tᵢ f + ħ Tᵢ ∂_Tᵢ f;
        D_α[f_] := ∂_α f + γ 𝒜 ∂_𝒜 f; D_αᵢ_[f_] := ∂_αᵢ f + γ 𝒜ᵢ ∂_𝒜ᵢ 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

```
(*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[1+Length[{ξs}]][
  (collect[P//Zip_{ξs},ξ] /. f_.ξ^d_.:>(D_{ξ*,d}[f]))/.ξ*→0/.((ξ*/.{b→B,t→T,α→𝒜})→1)]*)
```

Program

```
Zip_ξs_[Ps_List] := Zip_ξs /@ Ps;
Zip_ξs_[sd_SeriesData] := MapAt[Zip_ξs, sd, 3];
Zip_ξs_List[P_] := PP_Zip@Module[{zs, Zs},
   zs = Table[ξ*, {ξ, ξs}]; Zs = zs /. {b → B, t → T, α → 𝒜};
   CF@Total[
     CoefficientRules[P, ξs] /.
      (ps_ → c_) :> (D_Thread@{zs,ps}[c] /. Alternatives@@zs → 0 /. Alternatives@@Zs → 1)
    ]]
```

Program

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

```
$QZipFail = False;
QZip{}[E_,𝔼] := E;
QZipξs_List@𝔼[V_, L_, Q_, P_] :=
  PPQZip@Module[{ξ, z, zs, c, ys, ηs, qt, zrule, ξrule, out},
    zs = Table[ξ*, {ξ, ξs}];
    c = CF[Q /. Alternatives @@ (ξs ⋃ zs) → 0];
    ys = CF@Table[∂ξ (Q /. Alternatives @@ zs → 0), {ξ, ξs}];
    ηs = CF@Table[∂z (Q /. Alternatives @@ ξs → 0), {z, zs}];
    qt = CF@Inverse@Table[Kδz,ξ* - ∂z,ξQ, {ξ, ξs}, {z, zs}];
    zrule = Thread[zs → CF[qt.(zs + ys)]];
    ξrule = Thread[ξs → ξs + ηs.qt];
    out = CF /@ 𝔼[Complement[V, ξs],
       L, c + ηs.qt.ys, Det[qt] Zipξs[P /. (zrule ⋃ ξrule)]];
    If[¬ ($QZipFail ∨ TrueQ[out ≡ 𝔼3@QZipξs@𝔼4@𝔼[V, L, Q, P]]),
     $QZipFail = True; Print["QZip4 fail at {L,Q,P}=", {L, Q, P}]
    ];
    out
   ];
QZip@𝔼[V_, 𝒪s___] := QZipCases[V,(ξ|y)_]@𝔼[V, 𝒪s];
```

Program

```
$QZipFail = False;
QZipξs_List@𝔼[V_, ω_, L_, Q_, Ps_] :=
  PPQZip4@Module[{ξ, z, zs, c, ys, ηs, qt, zrule, ξrule},
    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[Kδz,ξ* - ∂z,ξQ, {ξ, ξs}, {z, zs}];
    zrule = Thread[zs → CF[qt.(zs + ys)]];
    ξrule = Thread[ξs → ξs + ηs.qt];
    CF /@
     𝔼[Complement[V, ξs], ω Det[qt / ω], L, c + ηs.qt.ys, Zipξs[Ps /. (zrule ⋃ ξrule)]] ];
```

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 $\mathbb{E}(L, Q, P) = \mathrm{P}e^{L+Q}$. 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.

Program

```
LZip{}[E_, 𝔼] := E;
LZip_ζs_List@𝔼[V_, L_, Q_, P_] :=
  PP_LZip@Module[{ζ, z, zs, Zs, c, ys, ηs, lt, zrule, Zrule, ζrule, Q1, EEQ, EQ},
    zs = Table[ζ*, {ζ, ζs}];
    Zs = zs /. {b → B, t → T, α → 𝒜};
    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~Join~
      (zrule /. r_Rule :↦ ((U = r[[1]] /. {b → B, t → T, α → 𝒜}) → (U /. U2l /. r //. l2U)));
    ζrule = Thread[ζs → ζs + ηs.lt];
    Q1 = Q /. (Zrule ⋃ ζrule);
    EEQ[ps___] := EEQ[ps] = (PP_"EEQ"@(CF[e^(-Q1) D_Thread[{zs,{ps}}][e^Q1]] /.
        {Alternatives@@zs → 0, Alternatives@@Zs → 1}));
    CF /@ ((*CF/@*)𝔼[Complement[V, ζs],
      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)
    ])
  ];
LZip@𝔼[V_, ℰs___] := LZip_Cases[V, (β|τ|a)_]@𝔼[V, ℰs];
```

Program

```
Zip[E_, 𝔼] := E // LZip // QZip;
```

Program

```
B{}[L_, R_] := L R;
B{is__}[L_, 𝔼, R_, 𝔼] := PP_B@Module[{n},
    MapAt[(# ⋃ Join@@Table[{β_(n@i), τ_(n@i), a_(n@i), ξ_(n@i), y_(n@i)}, {i, {is}}]) &,
     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}}]
     ], 1]];
B_is___[L_, R_] := B{is}[L, R];
```

Program

𝔼 morphisms with domain and range.

Program

```
𝔼sp___[L : Except[_List | _Pattern], 𝒪s___] := 𝔼sp[{}, L, 𝒪s];
Bis_List[𝔼d1_→r1_[V1_, L1_, Q1_, P1_], 𝔼d2_→r2_[V2_, L2_, Q2_, P2_]] :=
  𝔼(d1⋃Complement[d2,is])→(r2⋃Complement[r1,is]) @@ Bis[𝔼[V1, L1, Q1, P1], 𝔼[V2, L2, Q2, P2]];
𝔼d1_→r1_[V1_, L1_, Q1_, P1_] // 𝔼d2_→r2_[V2_, L2_, Q2_, P2_] :=
   Br1⋂d2[𝔼d1→r1[V1, L1, Q1, P1], 𝔼d2→r2[V2, L2, Q2, P2]];
𝔼d1_→r1_[V1_, L1_, Q1_, P1_] ≡ 𝔼d2_→r2_[V2_, L2_, Q2_, P2_] ^:=
   (d1 == d2) ∧ (r1 == r2) ∧ (𝔼[V1, L1, Q1, P1] ≡ 𝔼[V2, L2, Q2, P2]);
𝔼d1_→r1_[V1_, L1_, Q1_, P1_] 𝔼d2_→r2_[V2_, L2_, Q2_, P2_] ^:=
   𝔼(d1⋃d2)→(r1⋃r2) @@ (𝔼[V1, L1, Q1, P1] 𝔼[V2, L2, Q2, P2]);
𝔼d_→r_[V_, L_, Q_, P_]$k_ := 𝔼d→r @@ (𝔼[V, L, Q, P]$k);
Zip@𝔼d_→r_[V_, L_, Q_, P_] := 𝔼d→r @@ Zip@𝔼[V, L, Q, P];
𝔼_[𝒪___][i_] := {𝒪}[[i]];
```