Pensieve header: The "Speedy" engine sans γ.

Program

The "Speedy" Engine

Program

Internal Utilities

Program

Canonical Form:

Program

```
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 | 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

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

Upper to lower and lower to Upper:

Program

```
U2l = {B_i^(p_.) :> e^(-p ħ b_i), B^(p_.) :> e^(-p ħ b), T_i^(p_.) :> e^(p ħ t_i), T^(p_.) :> e^(p ħ t), 𝒜_i^(p_.) :> e^(p α_i), 𝒜^(p_.) :> e^(p α)};
l2U = {e^(c_. b_i+d_.) :> B_i^(-c/ħ) e^d, e^(c_. b+d_.) :> B^(-c/ħ) 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 e^d, e^(c_. α+d_.) :> 𝒜^c e^d,
    e^(ℰ_) :> e^(Expand@ℰ)};
```

Program

Derivatives in the presence of exponentiated variables:

Program

```
D_b[f_] := ∂_b f - ħ B ∂_B f; D_(b_i)[f_] := ∂_(b_i) f - ħ 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 + 𝒜 ∂_𝒜 f; D_(α_i)[f_] := ∂_(α_i) f + 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

```
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}) → 1))
```

Program

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.

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

Program

```
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[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[L, c + ηs.qt.ys, Det[qt] Zip_ζs[P /. (zrule ⋃ ζrule)]] ];
```

Program

LZip implements the "L-level zips" on $\mathbb{E}(L, Q, P) = Pe^{L+Q}$. Such zips regard all of Pe^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_] :=
  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 /. Alternatives @@ Zs → 1;
   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 = Join[zrule,
     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] =
     (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_ξs[(EQ @@ zs) (P /. (Zrule ⋃ ξrule))] /.
         Derivative[ps___][EQ][___] :> EEQ[ps] /. _EQ → 1)]];
```

Program

```
B_{}[L_, R_] := L R;
B_{is__}[L_𝔼, R_𝔼] := 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[{ξ_{n@i}, y_{n@i}}, {i, {is}}]];
B_is___[L_, R_] := B_{is}[L, R];
```

Program

𝔼 morphisms with domain and range.

Program

```
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_dr_[L_, Q_, P_]$k_ := 𝔼_dr @@ 𝔼[L, Q, P]$k;
𝔼_[ℰ___][i_Integer] := {ℰ}[[i]];
```

Program

$\mathbb{E}[\Lambda]$

Program

```
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]
```

Program

Exponentials as needed.

Program

Task. Define $\mathrm{Exp}_{m,i,k}[P]$ to compute $e^{\mathcal{O}(P)}$ to ϵ^k in the using the $m_{i,i\to i}$ multiplication, where P is an ϵ-dependent near-docile element, giving the answer in $\mathbb{E}$-form.

Methodology. If $P_0 := P_{\epsilon=0}$ and $e^{\lambda\,\mathcal{O}(P)} = \mathcal{O}\left(e^{\lambda P_0}\,F(\lambda)\right)$, then $F(\lambda=0)=1$ and we have:

$$\mathcal{O}\left(e^{\lambda P_0}(P_0\,F(\lambda)+\partial_\lambda F)\right) = \mathcal{O}\left(\partial_\lambda e^{\lambda P_0}\,F(\lambda)\right) = \partial_\lambda\,\mathcal{O}\left(e^{\lambda P_0}\,F(\lambda)\right) = \partial_\lambda e^{\lambda\,\mathcal{O}(P)} = e^{\lambda\,\mathcal{O}(P)}\,\mathcal{O}(P) = \mathcal{O}\left(e^{\lambda P_0}\,F(\lambda)\right)\mathcal{O}(P).$$

This is a linear ODE for F. Setting inductively $F_k = F_{k-1} + \epsilon^k\,\varphi$ we find that $F_0 = 1$ and solve for φ.

Program

```
(* Bug: The first line is valid only if 𝒪(e^P0) == e^𝒪(P0). *)
Exp_m_,i_,0[P_] := Module[{LQ = Normal@P /. ϵ → 0},
   𝔼[LQ /. (x | y)_i → 0, LQ /. (b | a | t)_i → 0, 1]];
```

Program

```
Exp_m_,i_,k_[P_] := Block[{$k = k},
  Module[{P0, λ, φ, φs, F, j, rhs, eqn, pows, at0, atλ},
   P0 = Normal@P /. ϵ → 0;
   F = Normal@Last@Exp_m,i,k-1[λ P];
   While[
    rhs = m_i,j→i[𝔼_{}→{i}[λ P0 /. (x | y)_i → 0, λ P0 /. (b | a | t)_i → 0, F]_k
          sσ_i→j@𝔼_{}→{i}[0, 0, P]_k] // Last // Normal;
    eqn = CF[(∂_λ F) + P0 F - rhs];
    eqn =!= 0, (*do*)
    pows = First /@ CoefficientRules[eqn, {y_i, b_i, a_i, x_i}];
    F += Sum[ϵ^k φ_js[λ] Times @@ {y_i, b_i, a_i, x_i}^js, {js, pows}];
    rhs = m_i,j→i[𝔼_{}→{i}[λ P0 /. (x | y)_i → 0, λ P0 /. (b | a | t)_i → 0, F]_k
          sσ_i→j@𝔼_{}→{i}[0, 0, P]_k] // Last // Normal;
    eqn = CF[(∂_λ F) + P0 F - rhs];
    φs = Table[φ_js[λ], {js, pows}];
    at0 = Table[φ_js[0] == 0, {js, pows}];
    atλ = (# == 0) & /@ (pows /. CoefficientRules[eqn, {y_i, b_i, a_i, x_i}]);
    F = F /. DSolve[And @@ (at0 ⋃ atλ), φs, λ][[1]]
   ];
   𝔼_{}→{i}[P0 /. (x | y)_i → 0, P0 /. (b | a | t)_i → 0, F + O[ϵ]^(k+1) /. λ → 1]]]
```

Program

"Define" Code

Program

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.

Program

```
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[op_nisp,$k_Integer, Block[{i, j, k}, op_isp,$k = ℰ; op_nis,$k]];
      SD[op_isp, op_{is},$k]; SD[op_sis__, 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_}
     }] ]]
```