

```

SWxy[U_, kk_] :=
SWxy[U, kk] = Block[{ $U = U, $k = kk, $p = kk },
Module[{ G, F, fs, f, bs, e, b, es },
G = Simp[Table[ξk/k!, {k, 0, $k+1}].
NestList[Simp[B[xU, #]] &, yU, $k+1]];
fs = Flatten@Table[f1,i,j,k[η], {1, 0, $k}, {i, 0, 1},
{j, 0, 1}, {k, 0, 1}];
F = fs.(bs = fs /. f1,i,j,k[η] => ei U @ {yi, aj, xk});
es = Flatten[Table[Coefficient[e, b] == 0,
{e, {F - 1U /. η → 0, F ** G - yU ** F - ∂ηF}},
{b, bs}]]];
F = F /. DSolve[es, fs, η][[1]];
IE[0,
ξ x + η y + (U /. {CU → -t η ξ, QU → η ξ (1 - T) / h}),
F + 0$k /. {e → 1, U → Times}
] /. (v : η | ξ | t | T | y | a | x) → v1
]];
tSWxy, i_, j → k_ :=
SWxy[$U, $k] /. {ξ1 → ξi, η1 → ηj, (v : t | T | y | a | x)1 → vk};
tSWxa, i_, j → k_ := IE[αj ak, e-Y αj ξi xk, 1];
tSWay, i_, j → k_ := IE[αi ak, e-Y αi ηj yk, 1];

```

## Exponentials as needed.

Task. Define  $\text{Exp}_{U_i,k}[\xi, P]$  which computes  $e^{\xi \mathcal{O}(P)}$  to  $\epsilon^k$  in the algebra  $U_i$ , where  $\xi$  is a scalar,  $X$  is  $x_i$  or  $y_i$ , and  $P$  is an  $\epsilon$ -dependent near-docile element, giving the answer in  $\mathbb{E}$ -form. Should satisfy  $U @ \text{Exp}_{U_i,k}[\xi, P] == \mathbb{S}_U[e^{\xi X}, x \rightarrow \mathcal{O}(P)]$ .

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

$$\mathcal{O}(e^{\xi P_0} (P_0 F(\xi) + \partial_\xi F)) = \mathcal{O}(\partial_\xi e^{\xi P_0} F(\xi)) = e^{\xi P_0} \mathcal{O}(P) = \mathcal{O}(e^{\xi P_0} F(\xi)) \mathcal{O}(P)$$

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

```

(* Bug: The first line is valid only if O(e^{P_0}) == e^{O(P_0)}. *)
(* Bug: ξ must be a symbol. *)
ExpU, i_, 0[ξ_, P_] := Module[{LQ = Normal@P /. e → 0},
IE[ξ LQ /. (x | y)i → 0, ξ LQ /. (t | a)i → 0, 1] ];
ExpU, i_, k[ξ_, P_] := Block[{ $U = U, $k = k },
Module[{ P0, φ, φs, F, j, rhs, at0, atξ },
P0 = Normal@P /. e → 0;
φs = Flatten@Table[φj1,j2,j3[ξ], {j2, 0, k},
{j1, 0, 2 k + 1 - j2}, {j3, 0, 2 k + 1 - j2 - j1}];
F = Normal@Last@ExpU, i_, k-1[ξ, P] +
ek φs.(φs /. φjs[ξ] => Times @@ {yi, ai, xi}{js});
rhs =
Normal@
Last@
mi,j → i[IE[ξ P0 /. (x | y)i → 0, ξ P0 /. (t | a)i → 0, F + 0$k]
mi → j@IE[0, 0, P + 0$k]];
at0 = (# == 0) & /@
Flatten@CoefficientList[F - 1 /. ξ → 0, {yi, ai, xi});
atξ = (# == 0) & /@
Flatten@CoefficientList[(∂ξF) + P0 F - rhs,
{yi, ai, xi});
IE[ξ P0 /. (x | y)i → 0, ξ P0 /. (t | a)i → 0, F + 0$k] /.
DSolve[And @@ (at0 ∪ atξ), φs, ξ][[1]] ] ]

```

## Zip and Bind

```

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];
{t*, y*, a*, x*, z*} = {τ, η, α, ξ, ζ};
{τ*, η*, α*, ξ*, ζ*} = {t, y, a, x, z};
(u-i)* := (u*)i;
Zip[ ] [P_] := P;
Zip[ξ_, ξs_] [P_] :=
(Expand[P // Zip[ξs]] /. f-. ξd => ∂{ξ*, d} f) /. ξ* → 0
QZip implements the “Q-level zips” on  $\mathbb{E}(L, Q, P) = \text{Pe}^{L+Q}$ . Such zips regard the  $L$  variables as scalars.
QZipξs_List, simp@E[L_, Q_, P_] :=
Module[{ξ, z, zs, c, ys, ηs, qt, zrule, Q1, Q2},
zs = Table[ξ*, {ξ, ξs}];
c = Q /. Alternatives @@ (ξs ∪ zs) → 0;
ys = Table[∂ξ (Q /. Alternatives @@ zs → 0), {ξ, ξs}];
ηs = Table[∂z (Q /. Alternatives @@ ξs → 0), {z, zs}];
qt = Inverse@Table[Kδz, ξ* - ∂z, ξ Q, {ξ, ξs}, {z, zs}];
zrule = Thread[zs → qt. (zs + ys)];
Q2 = (Q1 = c + ηs.zs /. zrule) /. Alternatives @@ zs → 0;
simp /@ E[L, Q2, Det[qt] e-Q2 Zipξs[eQ1 (P /. zrule)]] ];
QZipξs_List := QZipξs, CF;

```

LZip implements the “L-level zips” on  $\mathbb{E}(L, Q, P) = \text{Pe}^{L+Q}$ . Such zips regard all of  $\text{Pe}^Q$  as a single “P”. Here the z’s are  $t$  and  $\alpha$  and the ξ’s are  $\tau$  and  $a$ .

```

LZipξs_List, simp@E[L_, Q_, P_] :=
Module[{ξ, z, zs, c, ys, ηs, lt, zrule, L1, L2, Q1, Q2},
zs = Table[ξ*, {ξ, ξs}];
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[Kδz, ξ* - ∂z, ξ L, {ξ, ξs}, {z, zs}];
zrule = Thread[zs → lt. (zs + ys)];
L2 = (L1 = c + ηs.zs /. zrule) /. Alternatives @@ zs → 0;
Q2 = (Q1 = Q /. T2t /. zrule) /. Alternatives @@ zs → 0;
simp /@
E[L2, Q2, Det[lt] e-L2-Q2
Zipξs[eL1+Q1 (P /. T2t /. zrule)]] // T2t ];
LZipξs_List := LZipξs, CF;
Bind[ ] [L_, R_] := L R;
Bind[is_] [L_E, R_E] := Module[{n},
Times[
L /. Table[(v : T | t | a | x | y)i → vnei, {i, {is}}],
R /. Table[(v : τ | α | ξ | η)i → vnei, {i, {is}}]
] // LZipFlatten@Table[{εnei, anei}, {i, {is}}] //
QZipFlatten@Table[{εnei, ynei}, {i, {is}}] ];
BL_List := BindL; Bis_ := Bind[is];
Bind[L_E] := ξ;
Bind[Ls_, ξs_List, R_] := Bindξs[Bind[Ls], R];

```

## Tensorial Representations

```

tη = t1 = IE[0, 0, 1 + 0$k];
tmi, j → k := Module[{tk},
IE[(τi + τj) tk + αi ak + αj ak, ηi yk + ξj xk, 1]
(tSWxy, i, j → tk) /. {ttk → tk, Ttk → Tk, ytk → e-Y αi yk,
atk → ak, xtk → e-Y αj xk});
mj → k [L_E] := ξ ~ Bj, k ~ tmj, k → k;
tm1, 2 → 3

```