Pensieve Header: The Algebra of Emergent Chord Diagrams.

Goal: Implement $\mathcal{O}_{red,ps:\{\},ss:\{\}}\Big[\mathcal{A}_0[\prod_{s\in ss} AW_s[\,..]]$,
$+\sum_{s1\le s2}\mathcal{A}_{c[s1,s2]}\Big[\prod_{s\in ss\cup\{\overline{s1},\overline{s2}\}} AW_s[\,..]\Big]\Big]$
including $\otimes$, $m_{i,j\to k}$ (only if $\{i,\ j\}$ are neighbors), $\mathcal{O}_{ss}$, CF (Canonical Form) and HCF (HOMFLYPT Canonical Form).

In[•]:=
```
SetDirectory["C:\\drorbn\\AcademicPensieve\\People\\Kuno"];
<< FreeLie.m
<< AwCalculus.m
<< FAA.m
```

```
FreeLie` implements / extends
 {*, +, **, $SeriesShowDegree, ⟨⟩, ∫, ≡, ad, Ad, adSeries, AllCyclicWords, AllLyndonWords,
  AllWords, Arbitrator, AS, ASeries, AW, b, BCH, BooleanSequence, BracketForm, BS, CC, Crop,
  cw, CW, CWS, CWSeries, D, Deg, DegreeScale, DerivationSeries, div, DK, DKS, DKSeries, EulerE,
  Exp, Inverse, j, J, JA, LieDerivation, LieMorphism, LieSeries, LS, LW, LyndonFactorization,
  Morphism, New, RandomCWSeries, Randomizer, RandomLieSeries, RC, SeriesSolve, Support,
  t, tb, TopBracketForm, tr, UndeterminedCoefficients, αMap, Γ, ι, Λ, σ, ħ, ⟌, ⟍}.

FreeLie` is in the public domain. Dror Bar-Natan is committed
  to support it within reason until July 15, 2022. This is version 150814.

AwCalculus` implements / extends {*, **, ≡, dA, dc, deg, dm, dS, dΔ, dη, dσ, El, Es, hA,
  hm, hS, hΔ, hη, hσ, RandomElSeries, RandomEsSeries, tA, tha, tm, tS, tΔ, tη, tσ, Γ, Λ}.

AwCalculus` is in the public domain. Dror Bar-Natan is committed
  to support it within reason until July 15, 2022. This is version 150909.
```

Bases

```
Basis_d_[O_AR,ps_,ss_] := PP_EMBasis[O_AR,ps,ss /@ Flatten[{
      A_0 /@ Basis_d,ps[Product[AW_s, {s, ss}]],
      Table[
       A_c[ss⟦i⟧] /@ (Basis_d-1,ps[Product[AW_s, {s, ss}] AW_ss⟦i⟧‾] AW_ss⟦i⟧~~[]), {i, Length[ss]}],
      Table[A_c[ss⟦i⟧,ss⟦j⟧] /@ (Basis_d-1,ps[Product[AW_s, {s, ss}] AW_ss⟦i⟧‾] AW_ss⟦j⟧‾[]),
       {i, Length[ss] - 1}, {j, i + 1, Length@ss}]
     }]]
```

```
Basis_d_[O_HR,ps_,ss_] := PP_EMBasis[O_HR,ps,ss /@ Flatten[{
      A_0 /@ Basis_d,ps[Product[AW_s, {s, ss}]],
      Table[
       A_c[ss⟦i⟧] /@ (Basis_d-1,ps[Product[AW_s, {s, ss}] CW_ss⟦i⟧‾] AW_ss⟦i⟧~~[]), {i, Length[ss]}],
      Table[A_c[ss⟦i⟧,ss⟦j⟧] /@ (Basis_d-1,ps[Product[AW_s, {s, ss}]] AW_ss⟦i⟧‾[] AW_ss⟦j⟧‾[]),
       {i, Length[ss] - 1}, {j, i + 1, Length@ss}]
     }]]
```

Utilities

```
𝒜a_[A1_] + 𝒜a_[A2_] ^:= 𝒜a[A1 + A2];
c_ * 𝒜a_[A_] /; FreeQ[c, 𝒜] ^:= 𝒜a[Expand[c A]];
𝒜_[0] = 0;
```

In[ ]:=
```
𝒪red_,ps_,ss_[x_] + 𝒪red_,ps_,ss_[y_] ^:= 𝒪red,ps,ss[x + y];
c_ * 𝒪red_,ps_,ss_[x_] ^:= 𝒪red,ps,ss[Expand[c x]];
𝒪__[0] = 0;
```

```
CF[𝒪red_,ps_,ss_[x_Plus]] := PPEMCF[𝒪red,ps,ss[red /@ x]];
CF[𝒪red_,ps_,ss_[x_]] := PPEMCF[𝒪red,ps,ss[red@x]]
```

AR: Reduction in A

```
AR[0] = 0;
AR[𝒜0[A_]] := 𝒜0[A];
AR[𝒜c[s_][A_]] :=
  PPEMAR@Module[{l, r}, 𝒜c[s][A // Δ~s~→l,r // m~s~,l→s~ // Δr→l,r // ms,r→s // Sl→l // ml,s~→s~ // η~s~]];
AR[𝒜c[s1_,s2_][A_]] := PPEMAR@Module[{l, r},
    𝒜c[s1,s2][A // Δ~s2~→l,r // ms2,r→s2 // Δl→l,r // ms1,r→s1 // Sl→l // ml,s1~→s1~ // η~s2~]];
```

HR: Reduction in the H Quotient

```
HR[0] = 0;
HR[𝒜0[A_]] := 𝒜0[A];
HR[𝒜c[s_][A_]] := PPEMHR@Module[{l, r}, 𝒜c[s][A // m~s~,s→s // tr~s~→s~ // η~s~]];
HR[𝒜c[s1_,s2_][A_]] := PPEMHR@𝒜c[s1,s2][A // ms1,s2~→s1 // ms2,s1~→s2 // η~s1~ // η~s2~];
```

Reordering strands

```
𝕆ss_[𝕆red_,ps_,s0s_[y_]] /; FreeQ[y, 𝒜0] := PP𝕆@CF@Module[{s1, s2},
    𝕆red,ps,ss[
     y /. 𝒜c[s1_,s2_][A1_] /;
        Position[ss, s1][[1, 1]] > Position[ss, s2][[1, 1]] :> red[𝒜c[s2,s1][A1]]
    ]];
𝕆ss_[𝕆red_,ps_,s0s_[𝒜0[A_] + y_.]] := PP𝕆@CF@Module[{i, j, s1, s2, u1, u2},
    𝕆red,ps,ss[Plus[
      𝒜0[A],
      y /. 𝒜c[s1_,s2_][A1_] /;
         Position[ss, s1][[1, 1]] > Position[ss, s2][[1, 1]] :> red[𝒜c[s2,s1][A1]],
      Sum[
       If[Position[s0s, s1 = ss[[i]]][[1, 1]] < Position[s0s, s2 = ss[[j]]][[1, 1]], 0,
        Sum[
         red[𝒜c[s1,s2][Expand[A (AWu1[p] AWu2[] - AWu1[] AWu2[p])] //
               𝔻[p]s1→s1,s1¯ // 𝔻[p]s2→s2,s2¯ // ms1,u1→s1 // mu2,s1¯→s1¯]],
         {p, ps}
        ]
       ],
       {i, Length[ss] - 1}, {j, i + 1, Length@ss}
      ]
     ]]];
𝕆[𝕆red_,ps_,ss_[ℰ_]] := 𝕆Sort[ss]@𝕆red,ps,ss[ℰ]
```

Pole Renaming

```
pσx_→y_[𝕆red_,ps_,ss_[ℰ_]] := PPEMpσ@𝕆red,Replace[x→y]/@ps,ss[ℰ // FA[x → y]]
```

Strand Renaming

```
sσi_→j_[𝕆red_,ps_,ss_[ℰ_]] :=
 PPEMsσ@𝕆red,ps,Replace[i→j]/@ss[ℰ /. 𝒜a_[A_] :> (𝒜Replace[i→j]/@a[A // σi→j // σi¯→j¯ // σĩ→j̃])]
```

Strand Multiplication

```
sm__[0] = 0;
𝔒red_,ps_,ss_[ℰ_] // sm_i_,j_→k_ := PPEMsm@
  𝔒@𝔒red,ps,{k}~Join~Complement[ss,{i,j}][First[𝔒red,ps,ss[ℰ] // 𝔒{i,j}~Join~Complement[ss,{i,j}]] /. {
      𝔄c[i,j][A_] :↦ 𝔄c[k][A // σ_j→k̃ // m_i,j→k̄ // σi→k],
      𝔄c[i][A_] :↦ 𝔄c[k][A // m_i,j→k̃ // σi→k // σ_i→k̄],
      𝔄c[j][A_] :↦ 𝔄c[k][A // mi,j→k // σ_j→k̄ // σ_j→k̃],
      𝔄c[i,x_][A_] :↦ 𝔄c[k,x][A // m_i,j→k̄ // σi→k],
      𝔄c[j,x_][A_] :↦ 𝔄c[k,x][A // mi,j→k // σ_j→k̄],
      𝔄a_[A_] :↦ 𝔄a[A // mi,j→k]
     }]
```

The Unit for Poles

In[•]:=
```
pη_x_[𝔒red_,ps_,ss_[ℰ_]] := 𝔒red,ps∪{x},ss[ℰ]
```

The Unit for Strands

In[•]:=
```
sη_i_[𝔒red_,ps_,ss_[ℰ_]] := 𝔒red,ps,ss~Join~{i}[ℰ /. 𝔄a_[A_] :↦ 𝔄a[A // ηi]]
```

Strand Doubling

```
sΔ_→_,_[𝕆red_,ps_,ss_[ℰ_]] := PPEMsΔ@Module[{u1, u2},
  𝕆@𝕆red,ps,Replace[i→Sequence[j,k]]/@ss[ℰ /. {
    𝒜0[A_] :> 𝒜0[A // Δi→j,k] + 𝒜c[j,k][Sum[Expand[A (AWu1[p] AWu2[] - AWu1[] AWu2[p])] //
                𝔻[p]i→i,ī // 𝔻[p]ī→ī,ĩ // Δi→j,k // Δī→j̄,k̄ // Δĩ→j̃,k̃ //
              mj,j̄→j // σj̃→j̄ // mk̄,k̃→k̄ // mj,u1→j // mu2,j̄→j̄, {p, ps}]],
    𝒜c[x_,i][A_] :>
     𝒜c[x,j][A // Δi→j,k // Δī→j̄,k̄ // mk,k̄→k] + 𝒜c[x,k][A // Δi→j,k // Δī→j̄,k̄ // mj,j̄→j],
    𝒜c[i,y_][A_] :>
     𝒜c[j,y][A // Δi→j,k // Δī→j̄,k̄ // mk,k̄→k] + 𝒜c[k,y][A // Δi→j,k // Δī→j̄,k̄ // mj,j̄→j],
    𝒜c[i][A_] :> 𝒜c[j][A // Δi→j,k // Δī→j̄,k̄ // Δĩ→j̃,k̃ // mk,k̄→k // mk,k̃→k] +
      𝒜c[k][A // Δi→j,k // Δī→j̄,k̄ // Δĩ→j̃,k̃ // mj,j̄→j // mj,j̃→j] +
      𝒜c[j,k][A // Δi→j,k // Δī→j̄,k̄ // Δĩ→j̃,k̃ // mj̄,j̃→j̄ // mk,k̄→k // σk̃→k̄] +
      𝒜c[j,k][A // Δi→j,k // Δī→j̄,k̄ // Δĩ→j̃,k̃ // mk̄,k̃→k̄ // mj,j̄→j // σj̃→j̄],
    𝒜a_[A_] :> 𝒜a[A // Δi→j,k]
   }]
 ]
```

Pole Doubling

```
pΔx_→y_,z_[𝕆red_,ps_,ss_[ℰ_]] :=
 PPEMpΔ@𝕆red,Replace[x→Sequence[y,z]]/@ps,ss[ℰ /. 𝒜a_[A_] :> 𝒜a[A // FA[x → y + z]]]
```

Pole to strand conversion

```
p2sx_→i_[𝕆red_,ps_,ss_[ℰ_]] := PPEMp2s@CF@𝕆red,DeleteCases[ps,x],{i}~Join~ss[ℰ /. {
    𝒜0[A_] :> 𝒜0[A // FA[x → 0] // ηi] + Sum[
       𝒜c[i,s][A // 𝔻[x]s,s̄ // FA[x → 0] // ηi // ηī],
       {s, ss}],
    𝒜c_[A_] :> 𝒜c[A // FA[x → 0]]
   }]
```

Exterior Multiplication

```
EMd_[Ored_,ps1_,ss1_[ℰ1_],Ored_,ps2_,ss2_[ℰ2_]] /; ss1⋂ss2 === {} :=
 PPEMEM@Ored,ps1⋃ps2,ss1~Join~ss2[Expand[ℰ1 * ℰ2] /. {
     𝒜0[A1_] 𝒜0[A2_] :↦ 𝒜0[EMd[A1, A2]],
     𝒜0[A1_] 𝒜c_[A2_] :↦ 𝒜c[EMd-1[A1, A2]],
     𝒜c1_[_] 𝒜c2_[_] → 0
    }
  ]
```

Interior Multiplication

```
IMd_[Ored_,ps_,ss1_[ℰ1_],Ored_,ps_,ss2_[ℰ2_]] /; Sort[ss1] === Sort[ss2] :=
  PPEMIM@Module[{ℰ, ν},
    ℰ = Ored,ps,ss1[ℰ2];
    Do[ℰ = ℰ // Sσs→ν[s], {s, ss2}];
    ℰ = EMd[Ored,ps,ss1[ℰ1], ℰ];
    Do[ℰ = ℰ // Sms,ν[s]→s, {s, ss1}];
    ℰ
   ];
IMd_[ℰ1_, ℰ2_, ℰ3__] := IMd[IMd[ℰ1, ℰ2], ℰ3]
```