Free Lie Algebras Routines

Pensieve header: A free-Lie calculator, pensieve://Projects/WKO4/ branch, continues pensieve://2014-01/. Now with “New”!

Global Definitions

```
$SeriesShowDegree = 3; $SeriesCompareDegree = 3;
```

“New”

```
SetAttributes[New, HoldAll];
(*New[type_[srn_], def_] := Module[   (* "srn" for "self-reference name" *)
   {un}, (* "un" for "unique name" *)
   un=Unique[ToString[type]<>"$"];
   ReleaseHold[Hold[def] /. srn → un];
   type[un]
  ];*)
New[type_[srn_], def_] := (  (* "srn" for "self-reference name" *)
   ReleaseHold[Hold[def; type[srn]] /. srn → Unique[ToString[type] <> "$"]]
  );
```

Words and Lyndon Words

A Lyndon word is a word lexicographically smaller than all of its proper right factors.

```
AllWords[n_, ab_List] := AW /@ StringJoin @@@ Tuples[ToString /@ ab, n];
LyndonQ[AW[w_String]] := And @@ (
    OrderedQ[{w, #}] & /@ Table[StringDrop[w, i], {i, 1, StringLength[w] - 1}]);
AllLyndonWords[n_Integer, ab_List] := AllLyndonWords[n, ab] =
   LW @@@ Select[AllWords[n, ab /. LW[w_String] :> w], LyndonQ];
AllLyndonWords[{n_}, ab_List] := Join @@ Table[AllLyndonWords[k, ab], {k, n}];
Deg[LW[x_]] := StringLength[x];
LyndonFactorization[w_LW /; Deg[w] == 1] := w;
LyndonFactorization[LW[w_String] /; Deg[LW@w] > 1] := Module[{rf},
   rf = First[Sort[Table[StringDrop[w, i], {i, 1, StringLength[w] - 1}]]];
   LW /@ {StringDrop[w, -StringLength[rf]], rf}];
LW[s_Symbol] := LW[ToString[s]];
LW[LW[w_]] := LW[w];
LW /: LW[x_] ≤ LW[y_] := OrderedQ[{x, y}];
LW /: x_LW ≥ y_LW := y ≤ x; LW /: x_LW > y_LW := ! (x ≤ y);
LW /: x_LW < y_LW := ! (y ≤ x);
BracketForm[w_LW] /; Deg[w] == 1 := w[[1]];
BracketForm[w_LW] := BracketForm[w] = StringJoin[Flatten[{
      "[", BracketForm /@ LyndonFactorization[w], "]"
     }]];
topbracketform[w_LW] /; Deg[w] == 1 := w[[1]];
topbracketform[w_LW] := topbracketform[w] = Overscript[
    Row[Riffle[topbracketform /@ LyndonFactorization[w], " "]], ⟌];
TopBracketForm[w_LW] /; Deg[w] == 1 := Overscript[w[[1]], ⟌];
TopBracketForm[w_LW] := topbracketform[w];
TopBracketForm[CW[w_String]] := Overscript[w, ⏜];
TopBracketForm[expr_] := expr /. w_LW | w_CW :> TopBracketForm[w];
LW[is__Integer] := LW[StringJoin @@
    (StringTake["0123456789abcdefghijklmnopqrstuvwxyz", {#}] & /@ (1 + {is}))];
```

The Bracket for Lie Elements

```
b[0, _] = 0; b[_, 0] = 0;
b[c_ * (x_AW | x_LW), y_] := Expand[c b[x, y]];
b[x_, c_ * (y_AW | y_LW)] := Expand[c b[x, y]];
b[x_Plus, y_] := b[#, y] & /@ x;
b[x_, y_Plus] := b[x, #] & /@ y;
b[w_LW, z_LW] := LWAdjoint[w][z];
ad[x_][y_] := b[x, y];
```

```
LWAdjoint[w_] := LWAdjoint[w] = Module[{u},
     u = Unique[LWAct];
     u[z_] := u[z] = Which[
         w === z, 0,
         z < w, Expand[-b[z, w]],
         Deg[w] == 1, LW[First[w] <> First[z]],
         True, Module[{x, y},
          {x, y} = LyndonFactorization[w];
          If[y ≥ z,
           LW[First[w] <> First[z]],
           b[x, LWAdjoint[y][z]] + b[LWAdjoint[x][z], y]
          ]
         ]
        ];
     u
    ];
```

LieSeries

```
LieSeries[ser_Symbol][{dd_Integer}] :=
  TopBracketForm[LS @@ Table[ser[d], {d, dd}]];
LieSeries[ser_Symbol][e___] := ser[e];
Format[s_LieSeries, StandardForm] := TopBracketForm[s[{$SeriesShowDegree}]];
ShowLieSeries[d_Integer][s_LieSeries] := s[{d}];
MakeLieSeries[s_LieSeries] := s;
MakeLieSeries[expr_] := MakeLieSeries[expr] = New[LieSeries[ser],
    ser[d_Integer] := ser[d] = Expand[expr /. w_LW /; Deg[w] ≠ d → 0]
   ];
MakeLieSeries[ab_List, coefs_] := New[LieSeries[ser],
   ser[d_Integer] := ser[d] =
     Plus @@ MapIndexed[((coefs @@ Prepend[#2, d]) * #1) &, AllLyndonWords[d, ab]]
  ];
s1_LieSeries ≡ s2_LieSeries := Module[{res = True, k},
   For[k = 1, res && k <= $SeriesCompareDegree, ++k, res = res && (s1[k] == s2[k])];
   res
  ];
Crop[s_LieSeries, d_Integer] := Crop[s, d] = New[LieSeries[ser],
    ser[dd_Integer] := If[dd ≤ d, s[dd], 0]
   ];
RandomLieSeries[ab_List, opts___Rule] := Module[
   {rand = Randomizer /. {opts} /.
      Randomizer → (RandomInteger[{-2 Deg[#]!, 2 Deg[#]!}] / Deg[#]! &)},
   New[LieSeries[ser],
    ser[d_Integer] := ser[d] = Plus @@ ((rand[#] * #) & /@ AllLyndonWords[d, ab])
   ]
  ];
```

```
AddLieSeries[ss___LieSeries] := AddLieSeries[ss] = New[LieSeries[ser],
    ser[d_Integer] := ser[d] = Plus @@ ((#[d]) & /@ {ss})
   ];
LieSeries /: Plus[ss__LieSeries] := AddLieSeries[ss];
ScaleLieSeries[c_, s_LieSeries] := ScaleLieSeries[c, s] = New[LieSeries[ser],
    ser[d_Integer] := ser[d] = Expand[c * s[d]]
   ];
LieSeries /: c_ * s_LieSeries := ScaleLieSeries[c, s];
IntegrateLieSeries[ls_LieSeries, {s_, s0_, s1_}] :=
  IntegrateLieSeries[ls, {s, s0, s1}] = New[
    LieSeries[ser],
    ser[d_Integer] := ser[d] = Expand[∫_s0^s1 ls[d] ds]
   ];
LieSeries /: Integrate[ls_LieSeries, {s_, s0_, s1_}] :=
  IntegrateLieSeries[ls, {s, s0, s1}];

b[s1_LieSeries, s2_LieSeries] := b[s1, s2] = New[LieSeries[ser],
    ser[d_Integer] := ser[d] = ∑_k=1^d-1 b[s1[k], s2[d - k]]
   ];
b[s_LieSeries, y_] := b[s, MakeLieSeries[y]];
b[x_, s_LieSeries] := b[MakeLieSeries[x], s];
```

EulerE, DegreeScale

```
LieSeries /: EulerE[s_LieSeries] := New[LieSeries[s][ser],
   ser[d_Integer] := ser[d] = Expand[d * s[d]]
  ];
DegreeScale[h_][s_LieSeries | s_CWSeries] := New[Head[s][ser],
   ser[d_Integer] := ser[d] = Expand[h^d s[d]]
  ];
```

adSeries, and Ad

Convention: ad(x)(z)=[x,z]. Satisfies ad([x,y])=[ad(x),ad(y)].

```
adSeries[1, x_LieSeries][ψ_LieSeries] := adSeries[1, x][ψ] = ψ;
adSeries[ad^n_-, x_LieSeries][ψ_LieSeries] :=
  adSeries[ad^n, x][ψ] = New[LieSeries[ser],
    ser[d_Integer] := ser[d] = b[x, adSeries[ad^n-1, x][ψ]][d]
   ];
adSeries[f_, x_LieSeries][ψ_LieSeries] :=
  adSeries[f, x][ψ] = New[LieSeries[ser],
    ser[d_Integer] := ser[d] = Module[{c},
       Expand[Sum[
         c = SeriesCoefficient[f, {ad, 0, k}];
         If[c === 0, 0, c * adSeries[ad^k, x][ψ][d]],
         {k, 0, d - 1}
        ]]
      ]
   ];
adSeries[f_, x_][ψ_] := adSeries[f, MakeLieSeries[x]][MakeLieSeries[ψ]];
Ad[x_] := adSeries[E^ad, x];
```

LieDerivation, DerivationPower, DerivationSeries

```
LieDerivation[der_][es___] := der[es];
LieDerivation[rules___Rule] := LieDerivation[{rules}];
LieDerivation[rules_List] := LieDerivation[rules] = New[LieDerivation[der],
    der[Support] = First /@ rules;
    (der[w_LW] /; Deg[w] == 1) :=
     (der[w] = MakeLieSeries[w /. Append[rules, _LW → 0]]);
    der[w_LW] := der[w] = Module[{x, y},
       {x, y} = LyndonFactorization[w];
       AddLieSeries[b[der[x], y], b[x, der[y]]]
      ];
    der[s_LieSeries] := der[s] = New[LieSeries[ser],
       ser[d_] := ser[d] = Sum[der[s[k]][d], {k, 1, d}]];
    der[as_ASeries] := der[as] = New[ASeries[ser],
       ser[d_] := ser[d] = Sum[
          Expand[as[k] /. AW[w_] :> Sum[
               NonCommutativeMultiply[
                AW[StringTake[w, j - 1]],
                ι[der[LW[StringTake[w, {j}]]][d - k + 1]],
                AW[StringDrop[w, j]]
               ],
               {j, k}
              ]],
           {k, 1, d}
          ]
      ];
    der[cws_CWSeries] := der[cws] = New[CWSeries[ser],
       ser[d_] := ser[d] = Sum[
          Expand[cws[k] /. CW[w_] :> Sum[
               tr[NonCommutativeMultiply[
                 AW[StringTake[w, j - 1]],
                 ι[der[LW[StringTake[w, {j}]]][d - k + 1]],
                 AW[StringDrop[w, j]]
                ]],
               {j, k}
              ]],
           {k, 1, d}
          ]
      ];
    der[expr_][d_] :=
     Expand[expr /. {w_LW :> der[w][d], s_LieSeries :> der[s][d]}]
   ];
```

```
LieDerivation /: Plus[ders__LieDerivation] := LieDerivation[Table[
     u → Total[#[u] & /@ {ders}],
     {u, Union @@ (#[Support] & /@ {ders})}
    ]];
LieDerivation /: c_ * der_LieDerivation := LieDerivation[Table[
     u → (c der[u]),
     {u, der@Support}
    ]];

DerivationPower[0, der_LieDerivation][ψ_LieSeries | ψ_CWSeries] :=
  DerivationPower[0, der][ψ] = New[Head[ψ][ser],
    ser[d_Integer] := ser[d] = ψ[d]
   ];
DerivationPower[n_Integer, der_LieDerivation][ψ_LieSeries | ψ_CWSeries] :=
  DerivationPower[n, x][ψ] = New[Head[ψ][ser],
    ser[d_Integer] := ser[d] =  der[DerivationPower[n - 1, der][ψ]][d]
   ];
DerivationSeries[__][0] = 0;
DerivationSeries[f_, ld_LieDerivation][ψ_LieSeries | ψ_CWSeries] :=
  DerivationSeries[f, ld][ψ] = New[Head[ψ][ser],
    ser[d_Integer] := ser[d] = Module[{c},
       Expand[Sum[
         c = SeriesCoefficient[f, {der, 0, k}];
         If[c == 0, 0, c * DerivationPower[k, ld][ψ][d]],
         {k, 0, d}
        ]]
      ]
   ];
DerivationExp[ld_LieDerivation] := DerivationSeries[E^der, ld];
```

LieMorphism

```
LieMorphism[mor_][es___] := mor[es];
LieMorphism[rules__Rule] := LieMorphism[{rules}];
LieMorphism[rules_List] := LieMorphism[rules] = New[LieMorphism[mor],
    mor[Support] = First /@ rules;
    (mor[w_LW] /; Deg[w] == 1) := (mor[w] = MakeLieSeries[w /. rules]);
    mor[w_LW] := (mor[w] = b @@ (mor /@ LyndonFactorization[w]));
    mor[AW[""]] = MakeASeries[AW[""]];
    (mor[AW[w_]] /; StringLength[w] == 1) :=
     (mor[AW[w]] = ι[MakeLieSeries[LW[w] /. rules]]);
    mor[AW[w_]] := mor[AW[w]] = Module[{w1, w2},
       w1 = StringTake[w, Floor[StringLength[w] / 2]];
       w2 = StringDrop[w, Floor[StringLength[w] / 2]];
       (mor[AW[w1]]) ** (mor[AW[w2]])
      ];
    mor[CW[w_]] := tr[mor[AW[w]]];
    mor[s_LieSeries] := mor[s] = New[LieSeries[ser],
       ser[d_] := ser[d] = Sum[mor[s[k]][d], {k, 1, d}]];
    mor[cws_CWSeries] := mor[cws] = New[CWSeries[ser],
       ser[d_] := ser[d] = Sum[mor[cws[k]][d], {k, 1, d}]];
    mor[expr_][d_] := Expand[expr /. (w_LW | w_AW | w_CW) :> mor[w][d]]
   ];

LieMorphism /: Inverse[mor_LieMorphism] := InvertLieMorphism[mor];
InvertLieMorphism[mor_LieMorphism] := InvertLieMorphism[mor] =
  LieMorphism[Table[
    u -> New[LieSeries[uimg],
      uimg[1] = u;
      uimg[d_Integer] /; d > 1 := uimg[d] = -Sum[(mor[uimg[k]][d]), {k, 1, d-1}]
     ],
    {u, mor[Support]}
   ]]
```

StableApply

```
StableApply[mor_LieMorphism, (type : (LieSeries | ASeries | CWSeries))[s_]] := (
    StableApply[mor, type[s]] = New[type[ser],
      ser[d_] := ser[d] = Nest[mor, type[s], d][d]
     ]
  );
```

BCH

```
BCHBase = New[LieSeries[bch],
    bch[1] = LW@"x" + LW@"y";
    bch[d_Integer] := bch[d] = Expand[Plus[
          adSeries[e^-ad, MakeLieSeries[LW@"y"]][MakeLieSeries[LW@"x"]][d],
          -adSeries[(1 - e^-ad)/ad - 1, LieSeries[bch]][EulerE[LieSeries[bch]]][d]
        ]/d]
  ];
BCH[x_, y_] := LieMorphism[{LW["x"] → x, LW["y"] → y}][BCHBase];
```

AW, ASeries, ι, σ

```
Unprotect[NonCommutativeMultiply];
x_ ** 0 = 0; 0 ** y_ = 0;
(c_ * x_AW) ** y_ := Expand[c (x ** y)];
x_ ** (c_ * y_AW) := Expand[c (x ** y)];
x_Plus ** y_ := (# ** y) & /@ x;
x_ ** y_Plus := (x ** #) & /@ y;
Deg[AW[w_]] := StringLength[w];
AW[AW[w_]] := AW[w];
AW[w1_String] ** AW[w2_String] := AW[w1 <> w2];
b[w_AW, z_AW] := w ** z - z ** w;

ASeries[ser_Symbol][{dd_Integer}] := AS @@ Table[ser[d], {d, 0, dd}];
ASeries[as_Symbol][es___] := as[es];
Format[s_ASeries, StandardForm] := s[{$SeriesShowDegree}];
MakeASeries[as_CWSeries] := as;
MakeASeries[expr_] := MakeASeries[expr] = New[ASeries[ser],
    ser[d_Integer] := ser[d] = Expand[expr /. w_AW /; Deg[w] ≠ d → 0]
   ];
(s1_ASeries ** s2_ASeries) := (s1 ** s2) = New[ASeries[ser],
    ser[d_Integer] := ser[d] = Sum[s1[k] ** s2[d - k], {k, 0, d}]];
```

```
ι[w_LW] /; Deg[w] == 1 := AW @@ w;
ι[w_LW] := ι[w] = b @@ (ι /@ LyndonFactorization[w]);
ι[expr_] := Expand[expr /. w_LW :> ι[w]];
ι[ls_LieSeries] := ι[ls] = New[ASeries[as],
    as[0] = 0;
    as[d_] /; d > 0 := as[d] = ι[ls[d]]
   ];

σ[y_LW, w_LW] /; Deg[y] == 1 := σ[y, w] = Which[
    y === w, AW[""],
    Deg[w] === 1, 0,
    True, Module[{w1, w2},
     {w1, w2} = LyndonFactorization[w];
     ι[w1] ** σ[y, w2] - ι[w2] ** σ[y, w1]
    ]
   ];
σ[y_, ls_LieSeries] := σ[y, ls] = New[ASeries[as],
    as[d_] := as[d] = σ[LW[y], ls[d + 1]]
   ];
σ[y_, expr_] := Expand[expr /. w_LW :> σ[LW[y], w]];
```

CW, CWSeries, tr, div

```
Deg[CW[w_]] := StringLength[w];
AllCyclicWords[d_Integer, ab_List] := AllCyclicWords[d, ab] =
   Union[tr[AW[StringJoin @@ #] & /@ Tuples[ToString /@ ab, d]]];
CWSeries[cws_Symbol][es___] := cws[es];
CWSeries[ser_Symbol][{dd_Integer}] :=
  TopBracketForm[CWS @@ Table[ser[d], {d, dd}]];
Format[s_CWSeries, StandardForm] := TopBracketForm[s[{$SeriesShowDegree}]];
MakeCWSeries[cws_CWSeries] := cws;
MakeCWSeries[expr_] := MakeCWSeries[expr] = New[CWSeries[ser],
    ser[d_Integer] := ser[d] = Expand[expr /. w_CW /; Deg[w] ≠ d → 0]
   ];
MakeCWSeries[ab_List, coefs_] := New[CWSeries[ser],
   ser[d_Integer] := ser[d] =
     Plus @@ MapIndexed[((coefs @@ Prepend[#2, d]) * #1) &, AllCyclicWords[d, ab]]
  ];
RandomCWSeries[ab_List, opts___Rule] := New[CWSeries[ser],
   Module[
    {rand = Randomizer /. {opts} /.
        Randomizer → (RandomInteger[{-2 Deg[#]!, 2 Deg[#]!}] / Deg[#]! &)},
    ser[d_Integer] := ser[d] = Plus @@ ((rand[#] * #) & /@ AllCyclicWords[d, ab])
   ]];
s1_CWSeries ≡ s2_CWSeries := Module[{res = True, k},
   For[k = 1, res && k <= $SeriesCompareDegree, ++k, res = res && (s1[k] == s2[k])];
   res
  ];
AddCWSeries[ss___CWSeries] := AddCWSeries[ss] = New[CWSeries[ser],
    ser[d_Integer] := ser[d] = Plus @@ ((#[d]) & /@ {ss})
   ];
CWSeries /: Plus[ss__CWSeries] := AddCWSeries[ss];
ScaleCWSeries[c_, s_CWSeries] := ScaleCWSeries[c, s] = New[CWSeries[ser],
    ser[d_Integer] := ser[d] = Expand[c * s[d]]
   ];
CWSeries /: c_ * s_CWSeries := ScaleCWSeries[c, s];
IntegrateCWSeries[cws_CWSeries, {s_, s0_, s1_}] :=
  IntegrateCWSeries[cws, {s, s0, s1}] = New[CWSeries[ser],
    ser[d_Integer] := ser[d] = Expand[∫_s0^s1 cws[d] ds]];
CWSeries /: Integrate[cws_CWSeries, {s_, s0_, s1_}] :=
  IntegrateCWSeries[cws, {s, s0, s1}];
```

```
tr[w_AW] := tr[w] = CW[RotateToMinimal @@ w];
tr[expr_] := expr /. aw_AW :> tr[aw];
tr[as_ASeries] := tr[as] = New[CWSeries[cws], cws[d_] := cws[d] = tr[as[d]]];

div[y_LW, w_LW] /; Deg[y] == 1 := div[y, w] = tr[(AW @@ y) ** σ[y, w]];
div[y_, ls_LieSeries] := div[y, ls] = New[CWSeries[cws],
    cws[d_] := cws[d] = div[LW[y], ls[d]]
   ];
div[y_, expr_] := Expand[expr /. w_LW :> div[LW[y], w]];
div_y_[expr_] := div[y, expr];
```

The Meta-Cocycle JA

```
JA[-1, ___] = MakeCWSeries[0];
JA[n_, y_LW, μ_LieSeries, ss_] := JA[n, y, μ, ss] = Module[
    {s, sμ, μs},
    sμ = ScaleLieSeries[s, μ];
    μs = StableApply[LieMorphism[{y → Ad[ScaleLieSeries[1, sμ]][LW[z]]}], μ];
    μs = μs // LieMorphism[{LW[z] → y}];
    IntegrateCWSeries[
     AddCWSeries[
      JA[n - 1, y, μ, s] // LieDerivation[{y → b[μs, y]}],
      div[y, μs]
     ],
     {s, 0, ss}
    ]
   ];
JA[y_LW, μ_LieSeries] := JA[y, μ] = Module[{s}, New[CWSeries[cws],
     cws[d_Integer] := cws[d] = JA[d - 1, y, μ, s][d] /. s → 1
    ]];
```

CC, RC, ad_u, J

```
CC[us : {___}, γs : {_LieSeries ...}] :=
  LieMorphism[MapThread[Function[{u, γ}, u → Ad[γ][u]], {us, γs}]];
CC[u_, γ_LieSeries] := CC[{u}, {γ}];
CC_u_[γ_] := CC[u, γ];
RC[us : {___}, γs : {_LieSeries ...}] := Inverse[CC[us, -γs]];
RC[u_, γ_LieSeries] := RC[{u}, {γ}];
RC_u_[γ_] := RC[u, γ];
ad[us : {___}, γs : {_LieSeries ...}] :=
  LieDerivation[MapThread[Function[{u, γ}, u → b[γ, u]], {us, γs}]];
ad[u_, γ_LieSeries] := ad[{u}, {γ}];
ad_u_[γ_] := ad[u, γ];

J[u_, γ_] := J[u, γ] = Module[{s}, ∫_0^1 (γ // RC_u[s γ] // div_u // CC_u[-s γ]) ds];

J_u_[γ_] := J[u, γ];
```