Free Lie Algebras Routines

Pensieve header: A basic free-Lie calculator (no series, few operations), branched from pensieve://Projects/WKO4/.

Words and Lyndon Words

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

```
AllWords[n_, ab_List] := AW @@@ Tuples[ab, n];
LW /: LW[] < LW[__] = True;
LW /: _LW < LW[] = False;
LW /: LW[x_, xs___] < LW[x_, ys___] := LW[xs] < LW[ys];
LW /: LW[x_, ___] < LW[y_, ___] /; (x =!= y) := OrderedQ[{x, y}];
LW /: x_LW > y_LW := y < x;
LW /: x_LW ≤ y_LW := ! (y < x);
LW /: x_LW ≥ y_LW := ! (x < y);
LyndonQ[w_AW] := And @@ (
    (LW @@ w < LW @@ #) & /@ Table[Drop[w, i], {i, 1, Length[w] - 1}]);
AllLyndonWords[n_Integer, ab_List] := AllLyndonWords[n, ab] =
   LW @@@ Select[AllWords[n, ab /. LW[w_] ⧴ w], LyndonQ];
LyndonFactorization[w_LW /; Length[w] == 1] := w;
LyndonFactorization[w_LW /; Length[w] > 1] := Module[{rf},
   rf = First[Sort[Table[Drop[w, i], {i, 1, Length[w] - 1}], Less]];
   LW /@ {Drop[w, -Length[rf]], rf}];
LW[w_LW] := w;
BracketForm[LW[c_]] := ToString[c];
BracketForm[w_LW] := BracketForm[w] = StringJoin[Flatten[{
      "[", BracketForm /@ LyndonFactorization[w], "]"
     }]];
BracketForm[expr_] := expr /. w_LW ⧴ BracketForm[w];
topbracketform[LW[c_]] := c;
topbracketform[w_LW] := topbracketform[w] = Overscript[
    Row[Riffle[topbracketform /@ LyndonFactorization[w], " "]], ⎵];
TopBracketForm[LW[c_]] := Overscript[c, ⎵];
TopBracketForm[w_LW] := topbracketform[w];
TopBracketForm[expr_] := expr /. w_LW ⧴ TopBracketForm[w];
Format[w_LW] := TopBracketForm[w];
```

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] := Which[
  w === z, 0,
  z < w, Expand[-b[z, w]],
  Length[w] == 1, Join[w, z],
  True, Module[{x, y},
   {x, y} = LyndonFactorization[w];
   If[y ≥ z,
    Join[w, z],
    b[x, b[y, z]] + b[b[x, z], y]
   ]
  ]
 ]
```

```
{x, y, z} = LW /@ {1, 2, 3}
```

$\{\overline{1}, \overline{2}, \overline{3}\}$

```
{b[x, y], b[y, x]}
```

$\{\overline{1\,2}, -\overline{1\,2}\}$

```
{t1 = b[x, b[y, z]], t2 = b[y, b[z, x]], t3 = b[z, b[x, y]], t1 + t2 + t3}
```

$\{\overline{1\,\overline{2\,3}}, \overline{\overline{1\,3}\,2}, -\overline{1\,\overline{2\,3}} - \overline{\overline{1\,3}\,2}, 0\}$

```
b[b[x, y], b[x, b[x, y]]]
```

$-\overline{\overline{1\,\overline{1\,2}}\,\overline{1\,2}}$

LieDerivation, LieMorphism

```
LieDerivation[der_Symbol, rules___Rule] := LieDerivation[der, {rules}];
LieDerivation[der_Symbol, rules_List] := (
   (der[w_LW] /; Length[w] == 1) := (der[w] = w /. Append[rules, _LW → 0]);
   der[w_LW] := der[w] = Module[{x, y},
      {x, y} = LyndonFactorization[w];
      b[der[x], y] + b[x, der[y]]
     ];
   der[expr_] := Expand[expr /. w_LW :> der[w]];
   der
  );
LieMorphism[mor_Symbol, rules___Rule] := LieMorphism[mor, {rules}];
LieMorphism[mor_Symbol, rules_List] := (
   (mor[w_LW] /; Length[w] == 1) := (mor[w] = w /. rules);
   mor[w_LW] := mor[w] = b @@ (mor /@ LyndonFactorization[w]);
   mor[expr_] := Expand[expr /. w_LW :> mor[w]];
   mor
  );
```

```
LieDerivation[d, x → b[x, y]]
```

d

```
{b[x, z], b[x, b[x, z]]} // d
```

{1 2 3 + 1 3 2, 1 1 2 3 + 1 1 3 2 + 1 2 1 3}

```
LieMorphism[m, x → b[x, y]]
```

m

```
{b[x, z], b[x, b[x, z]]} // m
```

{1 2 3 + 1 3 2, 1 2 1 2 3 + 1 2 1 3 2}