```
<< KnotTheory`
```

```
Loading KnotTheory` version of March 26, 2009, 12:14:8.1008.
Read more at http://katlas.org/wiki/KnotTheory.
```

```
BeginPackage["KnotTheory`"];

MultivariableAlexander::usage = "
MultivariableAlexander[L][t] returns the multivariable Alexander polynomial
of a link L as a function of the variable t[1], t[2], ..., t[c], where c
is the number of components of L. MultivariableAlexander[L, Program -> prog][t]
uses the program prog to perform the computation. The currently available
programs are \"MVA1\", written by Dan Carney in Toronto in the summer of 2005,
and the faster \"MVA2\" (default), written by Jana Archibald in Toronto in 2008-9.
";

MultivariableAlexander::about = "The multivariable Alexander program \"MVA1\" was
written by Dan Carney at the University of Toronto in the summer of 2005; \"MVA2\"
was written by Jana Archibald in Toronto in 2008-9.";

Options[MultivariableAlexander] = {Program → "MVA2"};

Begin["`MVA`"];

MultivariableAlexander[Link[n_, t_, k_]] :=
 (Needs["KnotTheory`MultivariableAlexander4Links` "];
  Unset[MultivariableAlexander[Link[n1_, t1_, k1_]]];
  MultivariableAlexander[Link[n, t, k]])

MultivariableAlexander[L_, opts___] := Module[
  {prog = (Program /. {opts} /. Options[MultivariableAlexander])},
  Switch[prog,
   "MVA1", KnotTheory`MVA1`MVA1[L],
   "MVA2", KnotTheory`MVA2`MVA2[L]
  ]
 ]

End[];

Begin["`MVA1`"];

MVA1[K_] /; Head[K] =!= BR := (MVA1[BR[K]])
```

```
MVA1[BR[NotAvailable]] := (error &)

MVA1[BR[1, {}]] := (1 &)

MVA1[BR[2, braidWord_List]] := (MVA1[BR[3, Append[braidWord, 2]]])

MVA1[BR[numberOfStrings_Integer, permutations_List]] /; numberOfStrings ≥ 3 :=
  MVA1[BR[numberOfStrings, permutations]] =
   Module[{data}, CreditMessage["The multivariable Alexander program \"MVA1\" was written
       by Dan Carney at the University of Toronto in the summer of 2005."];
    If[numberOfStrings > 2, Null, Return[error];, Return[error]];
    If[! Scan[If[Abs[#1] < numberOfStrings, Null, Return[False];]; &, permutations],
     Return[error];];
    data[braidWidth] = numberOfStrings;
    data[braidWord] = permutations;
    data[braidHead] = Range[numberOfStrings];
    MultivariableAlexanderInner[data]];

SetAttributes[{braidWidth, braidHead, braidTail, braidWord, knotComponent, error,
   strandMapping, components, numberOfComponents, variableName, polynomial}, Protected];

dbgPrint = False;
Dbg[seq__] := (If[dbgPrint, Print[seq]];)

MultivariableAlexanderInner[data_] :=
  Module[{temp}, Dbg[Unevaluated["Braid Word ", data[braidWord]]];
   Scan[data[braidTail, #] = #; &, data[braidHead]];
   Dbg[Unevaluated["Braid Tail ", data[braidTail, #] & /@ data[braidHead]]];
   Scan[(PermutationFunction[data, braidTail, #];
      Dbg[Unevaluated["Braid Tail ", data[braidTail, #] & /@ data[braidHead]]];) &,
    data[braidWord]];
   IdentifyElements[data];
   Dbg[Unevaluated["Strand Components ", data[components]]];
   Dbg[Unevaluated["Variables ",
     ReplaceAll[data[variableName, #] & /@ data[braidHead], knotComponent → "T"]]];
   FormColouredBurauMatrix[data];
   Dbg[Unevaluated["Burau ",
     ReplaceAll[Expand[data[burau]], knotComponent → "T"] // MatrixForm]];
   Dbg[Unevaluated["Divisor ", ReplaceAll[data[divisor], knotComponent → "T"] //
      MatrixForm]];
   temp = data[burau] - data[divisor] * IdentityMatrix[data[braidWidth] - 1];
   temp = Expand[temp];
   Dbg[Unevaluated["Matrix ", ReplaceAll[temp, knotComponent → "T"] // MatrixForm]];
```

```
    temp = Det[temp];
    Dbg[Unevaluated["Determinant ", ReplaceAll[temp, knotComponent → "T"]]];
    data[polynomial] = Expand[Simplify[Factor[temp] / Factor[CalculateDivisor[data]]]];
    Dbg[Unevaluated["Polynomial ", ReplaceAll[data[polynomial], knotComponent → "T"]]];
    CalculateOutput[data]];

PermutationFunction[data_, list_, j_Integer] := Module[{temp, i}, i = Abs[j];
    temp = data[list, i];
    data[list, i] = data[list, i + 1];
    data[list, i + 1] = temp;];

IdentifyElements[data_] := Module[{marked, strand, component},
    Scan[(data[strandMapping, data[braidTail, #]] = #;) &, data[braidHead]];
    Dbg[Unevaluated["Strand Mapping ", data[strandMapping, #] & /@ data[braidHead]]];
    data[components] = {};
    Scan[(If[marked[#] =!= True, component = {};
          strand = #;
          While[marked[strand] =!= True, marked[strand] = True;
           component = Append[component, strand];
           strand = data[strandMapping, strand];];
          data[components] = Append[data[components], component];];) &, data[braidHead]];
    data[numberOfComponents] = Length[data[components]];
    For[component = 1, component ≤ data[numberOfComponents],
     component++, Scan[(data[variableName, #] = knotComponent[component]) &,
       data[components][[component]]];];];

CalculateDivisor[data_] :=
  Module[{temp = 1}, Scan[(temp *= data[variableName, #]) &, data[braidHead]];
    temp = If[data[numberOfComponents] == 1, (1 - temp) / (1 - data[variableName, 1]), 1 - temp];
    Dbg[Unevaluated["Divisor ", ReplaceAll[temp, knotComponent → "T"]]];
    temp];

CalculateOutput[data_] :=
  Module[{temp = 1, temp2, comps, term1}, If[data[polynomial] == 0, Return[0 &]];
    Scan[(temp2 = knotComponent[#]^Exponent[data[polynomial], knotComponent[#], Min];
       If[temp2 =!= 0, temp *= temp2;]) &, comps = Range[data[numberOfComponents]]];
    temp = Expand[data[polynomial] / temp];
    comps = knotComponent /@ comps;
    temp = First[
      Sort[Flatten[({temp, -temp} /. Thread[Rule[comps, #]]) & /@ Permutations[comps]]]];
    (*If[Head[temp]===Plus,term1=First[temp],term1=temp];
    If[(term1/._knotComponent→1)<0,temp=Expand[-temp]];*)
```

```
    Function @@ {ReplaceAll[temp, knotComponent → #]}];

GetSubmatrix[row_Integer, variableIndex_Integer, data_] :=
  Module[{output, variable}, variable = data[variableName, variableIndex];
   output = IdentityMatrix[data[braidWidth] - 1];
   output[[row, row]] = -variable;
   If[row ≠ data[braidWidth] - 1, output[[row, row + 1]] = 1, Null];
   If[row ≠ 1, output[[row, row - 1]] = variable, Null];
   Dbg[Unevaluated["Submatrix ", ReplaceAll[output, knotComponent → "T"] // MatrixForm]];
   data[burau] = data[burau].output;];

GetSubmatrixInverse[row_Integer, variableIndex_Integer, data_] :=
  Module[{output, variable}, variable = data[variableName, variableIndex];
   data[divisor] = variable * data[divisor];
   output = variable * IdentityMatrix[data[braidWidth] - 1];
   output[[row, row]] = -1;
   If[row ≠ data[braidWidth] - 1, output[[row, row + 1]] = 1, Null];
   If[row ≠ 1, output[[row, row - 1]] = variable, Null];
   Dbg[Unevaluated["Submatrix ", ReplaceAll[output, knotComponent → "T"] // MatrixForm]];
   data[burau] = data[burau].output;];

FormColouredBurauMatrix[data_] := Module[{tempArray}, data[divisor] = 1;
   data[burau] = IdentityMatrix[data[braidWidth] - 1];
   Scan[(data[tempArray, #] = #;) &, data[braidHead]];
   Scan[(If[# < 0, GetSubmatrixInverse[-1 * #, data[tempArray, -1 * # + 1], data];,
       GetSubmatrix[#, data[tempArray, #], data];];
      PermutationFunction[data, tempArray, #];) &, data[braidWord]];];

End[];

Begin["`MVA2`"];

MVA2[PD[Loop[_]]] := (1 / (#[1] - 1)) &
MVA2[K_] /; Head[K] =!= PD := MVA2[PD[K]]

MVA2[pd_PD] := MVA2[pd] = Module[
   {l, mat, skel, pd1, G, t, arcs, path, i, j, k, M, emb, done, pd2, rot, place},
   CreditMessage[
    "The multivariable Alexander program \"MVA2\" was written by Jana Archibald
      at the University of Toronto in 2007-2008."];
   l = Length[pd];
   mat = Table[0, {2 * l}, {2 * l}];
```

```
skel = Skeleton[pd];
pd1 = List @@ pd;
G = Table[0, {2 * l}, {l}];
pd1 //. X[a_, b_, c_, d_] :> If[d == b + 1 || b - d > 1,
   {mat[[c, a]] = -t[b]; mat[[c, b]] = t[a] - 1;  mat[[c, c]] = 1},
   {mat[[c, a]] = -1; mat[[c, b]] = 1 - t[a] ; mat[[c, c]] = t[b]}
  ];
arcs = Times @@ pd /. {
    X[i_, j_, k_, l_] /; (l - j == 1 || j - l > 1) :> path[k] path[i] path[j, l],
    X[i_, j_, k_, l_] /; (j - l == 1 || l - j > 1) :> path[k] path[i] path[l, j],
    P[i_, j_] :> path[i, j]
   } //. {
   path[a__, i_] path[i_, b__] :> path[a, i, b],
   path[a__, i_] path[b__, i_] :> Join[path[a, i], Reverse[path[b]]],
   path[i_, a__] path[i_, b__] :> Join[Reverse[path[b]], path[i, a]],
   path[a__, i_] path[i_] :> path[a, i],
   path[i_, a__] path[i_] :> path[a, i],
   path[i_] path[i_] :> path[i]
  };
If[Length[arcs] === l, For[i = 1, i ≤ 2 * l, i++,
  G = ReplacePart[G, 1, {i, First[First[Position[arcs, i]]]}]
 ]];
mat = mat /. t[a_] :> t[Position[skel, a][[1, 1]]];
If[Length[arcs] === l,
 M = Factor[Simplify[
    Det[
      Delete[
       Transpose[Delete[
         Transpose[G].mat.G,
         Position[arcs, pd1[[1, 3]]][[1, 1]]
        ]],
       Position[arcs, pd1[[1, 3]]][[1, 1]]
       ]
     ] / (t[Position[skel, pd1[[1, 3]]][[1, 1]]] - 1)
   ]],
 M = 0];
emb = Table[Null, {Length[pd]}];
done = Table[Null, {2 * Length[pd]}];
emb[[1]] = 0;
pd2 = pd;
rot = Table[0, {Length[skel]}];
place[i_, a_] := Module[
```

```
      {ni, na, arc, dir, oparc},
      arc = pd2[[i, a]];
      {{ni, na}} = Complement[Position[pd2, arc], {{i, a}}];
      If[emb[[ni]] === Null,
       emb[[ni]] = 3 - a + emb[[i]];
       pd2[[ni]] = RotateLeft[pd1[[ni]], na - 1];
       place[ni, #] & /@ {2, 3, 4},
       (* Else *) oparc = RotateLeft[pd2[[i]], 2][[a]];
       If[done[[arc]] === Null,
        done[[arc]] = 1;
        dir = If[arc - oparc == 1 || arc - oparc < -1, 1, -1];
        rot[[Position[skel, arc][[1, 1]]]] += dir * (emb[[ni]] - emb[[i]] + a - na - 2)
       ]
      ]
     ];
    place[1, #] & /@ {1, 2, 3, 4};
    k = -rot / 4;
    For[j = 1, j ≤ l, j++,
     k = ReplacePart[k,
       -1 + k[[Position[skel, pd[[j, 2]]][[1, 1]]], Position[skel, pd[[j, 2]]][[1, 1]]]
    ];
    For[i = 1, i ≤ Length[k], i++,
     M *= t[i]^((1 / 2) * k[[i]])
    ];
    If[pd[[1, 4]] == pd[[1, 2]] + 1 || pd[[1, 2]] - pd[[1, 4]] > 1,
     M *= t[Position[skel, pd[[1, 1]]][[1, 1]]] * t[Position[skel, pd[[1, 2]]][[1, 1]]],
     M *= t[Position[skel, pd[[1, 1]]][[1, 1]]]
    ];
    Evaluate[M /. t → #] &
   ]

End[];
EndPackage[]

KOs = Select[AllLinks[], (Crossings[#] ≤ 11) &];

MultivariableAlexander[PD[#]][t] & /@ KOs
```

KnotTheory::loading : Loading precomputed data in PD4Links`.

KnotTheory::credits :

The multivariable Alexander program "MVA2" was written by Jana Archibald at the University of Toronto in 2007–2008.