

Pensieve header: A concise implementation of the FastKh algorithm.

```
<< KnotTheory`
Loading KnotTheory` version of February 5, 2013, 3:48:46.4762.
Read more at http://katlas.org/wiki/KnotTheory.

SetAttributes[{P, S}, Orderless];
dot /: dot[_]^{k_} := 0;
( $\sigma_S$ )[i_] := First@Cases[ $\sigma$ , P[i, j_]  $\Rightarrow$  j];

(* ECP for "Equivalence Class Projection" *)
ECP[ $\lambda$ _List] := Module[{ $\rho$ , ec},
  ec = Fold[
    ( $\rho$  = First /@ Position[#1, #2];
     Append[Delete[#1, List /@  $\rho$ ], Union@@ (#1[[ $\rho$ ]])]) &,
     $\lambda$ , Union @@  $\lambda$ 
  ] // SortBy[#, First] &;
  Union@@Replace[ec, c_  $\Rightarrow$  ((#  $\rightarrow$  First[c]) & /@ c), {1}]
];
ECP[ $\lambda_S$ ] := ECP[Join[ $\lambda$ ] /. S | P  $\rightarrow$  List];
(* ECP for "Equivalence Class Representatives" *)
ECR[ $\lambda$ _] := Union[Last /@ ECP[ $\lambda$ ]];

VCLaw[ $\beta_S$ ,  $\mu_S$ ,  $\tau_S$ ] := VCLaw[ $\beta$ ,  $\mu$ ,  $\tau$ ] = Module[
  {p, ins1, ins2, outs,  $\chi_S$ ,  $\chi$ , h, law1, law2, dec},
  p = ECP[ $\beta$ ,  $\mu$ ,  $\tau$ ];
  ins1 = ECR[ $\beta$ ,  $\mu$ ]; ins2 = ECR[ $\mu$ ,  $\tau$ ]; outs = ECR[ $\beta$ ,  $\tau$ ];
  Times @@ (h /@ Join[ins1, ins2, outs] /. p)
   $\chi_S$  =  $\frac{\text{Times @@ (h /@ (Last /@ p))}^{1/2}}{\text{PowerExpand}[(\text{Times @@ (h /@ (Last /@ p))}^{1/2})]}$ ;
  dec =  $\chi_S$  /. h[i_]^{x_}  $\Rightarrow$  (2 dot[i])^{(2-x)/2};
  dec *= Times @@ MapThread[If[#1 == #2, 1, dot[#1] + dot[#2]] &,
    {outs, outs /. p}];
  law1 = dot /@ ins1; law1 = Thread[law1  $\rightarrow$  (law1 /. p)];
  law2 = dot /@ ins2; law2 = Thread[law2  $\rightarrow$  (law2 /. p)];
  {law1, law2, Expand[dec]}
];
VC[Cob[ $\beta_S$ ,  $\mu_S$ , dots1_], Cob[ $\mu_S$ ,  $\tau_S$ , dots2_]] := Module[
  {law1, law2, dec},
  {law1, law2, dec} = VCLaw[ $\beta$ ,  $\mu$ ,  $\tau$ ];
  Cob[ $\beta$ ,  $\tau$ , Expand[dec * (dots1 /. law1) (dots2 /. law2)]]
];
m[i_, j_][ $\sigma_S$ ] := Which[
   $\sigma$ [i]  $\neq$  j, Append[DeleteCases[ $\sigma$ , P[i, _] | P[_ , j]], P[ $\sigma$ [i],  $\sigma$ [j]]],
   $\sigma$ [i] == j, DeleteCases[ $\sigma$ , P[i, j]]
];
m[i_, j_][Q[k_] *  $\sigma_S$ ] := m[i, j][ $\sigma$ ] * If[ $\sigma$ [i]  $\neq$  j, {Q[k]}, {Q[k+1], Q[k-1]}];
```

```

m[i_, j_][Cob[β_S, τ_S, dots_]] := Module[
  {nβ, nτ, p, ijdot, ndots, x},
  {nβ, nτ} = m[i, j] /@ {β, τ};
  p = ECP[β, τ];
  ijdot = dot[Min[i, j]];
  ndots = Which[
    β[i] ≠ j && τ[i] ≠ j, {{If[(i/.p) ≠ (j/.p), 1, dot[β[i]] + dot[τ[i]]]}},
    β[i] = j && τ[i] ≠ j, {{1, ijdot}},
    β[i] ≠ j && τ[i] = j, {{ijdot}, {1}},
    β[i] = j && τ[i] = j,  $\begin{pmatrix} ijdot & 0 \\ 1 & ijdot \end{pmatrix}$ 
  ];
  ];
  ndots = Expand[dots * ndots] /. dot[k_] =>
    dot[k /. {i → β[i], j → β[j]}] /. {i → τ[i], j → τ[j]} /. ECP[nβ, nτ];
  If[β[i] = j && τ[i] = j, Coefficient[ndots /. ijdot → x, x], ndots]
];

Q /: Q[i_] Q[j_] := Q[i+j];
S /: s1_S * s2_S := Join[s1, s2];

Kom /: Kom[chains_, ds_] * Cob[β_, τ_, 1] := Module[{L, ρ, nchains, d, k},
  L = Length[chains]; ρ_k_ := ρ_k = Length[chains[[k]]]; ρ_0 = ρ_{L+1} = 0;
  Kom[
    nchains = MapThread[Join, List @@@ {
      Append[chains /. σ_S => β σ, {}],
      Prepend[chains /. σ_S => τ σ, {}]
    }],
    Table[
      If[(ρ_k + ρ_{k-1}) (ρ_{k+1} + ρ_k) == 0, 0,
        (*else*) d = Table[0, {ρ_{k+1} + ρ_k}, {ρ_k + ρ_{k-1}}];
        If[k ≤ L && ρ_k ρ_{k+1} ≠ 0, d[[1 ;; ρ_{k+1}, 1 ;; ρ_k]] = ds[[k]]];
        If[k ≤ L && ρ_k ≠ 0, d[[ρ_{k+1} + 1 ;; ρ_{k+1} + ρ_k, 1 ;; ρ_k]] = (-1)^k IdentityMatrix[ρ_k]];
        If[k > 1 && ρ_{k-1} ρ_k ≠ 0, d[[ρ_{k+1} + 1 ;; ρ_{k+1} + ρ_k, ρ_k + 1 ;; ρ_k + ρ_{k-1}]] = ds[[k-1]]];
        d
      ], {k, L}
    ]
  ]
];

Kom /: Show[Kom[chains_, ds_]] :=
  MatrixForm[{ColumnForm /@ chains, MatrixForm /@ Append[ds, 0]}];

```

```

m[i_, j_][Kom[chains_, ds_]] := Kom[
  Flatten /@ Map[m[i, j], chains, {2}],
  Table[
    If[Length[chains][[k]] == 0 || Length[chains][[k+1]] == 0, 0,
      (* else *) Table[
        m[i, j][Cob[chains][[k, b]] /. _Q → 1,
          chains[[k+1, a]] /. _Q → 1, ds[[k, a, b]]],
        {a, Length[chains][[k+1]]}, {b, Length[chains][[k]]}
      ] // ArrayFlatten
    ],
    {k, Length[ds]}]
];

Contract[kom_Kom] := Module[{chains, ds, L, done,  $\phi$ ,  $\gamma\delta$ },
  {chains, ds} = List @@ kom;
  L = Length[ds];
  Do[ (* {k,L} *)
    done = False; While[! done,
      done = True;
      Do[ (* {a, Length[chains][[k+1]]}, {b, Length[chains][[k]]} *)
        If[NumberQ[ $\phi$  = ds[[k, a, b]] &&  $\phi \neq 0$  && chains[[k+1, a]] == chains[[k, b]],
          done = False;
          If[Length[chains][[k]] > 1 && Length[chains][[k+1]] > 1,
             $\gamma\delta$  = Table[
              VC[
                Cob[chains][[k, d]], chains[[k+1, a]], ds[[k, a, d]] /. _Q → 1,
                Cob[chains][[k, b]], chains[[k+1, c]], ds[[k, c, b]] /. _Q → 1
              ][[3]],
              {c, Length[chains][[k+1]]}, {d, Length[chains][[k]]}
            ];
            ds[[k]] = Expand[Drop[ds[[k]] -  $\phi^{-1} \gamma\delta$ , {a}, {b}]],
            (* else *) ds[[k]] = 0
          ];
          chains[[k]] = Drop[chains[[k]], {b}];
          chains[[k+1]] = Drop[chains[[k+1]], {a}];
          If[k > 1, ds[[k-1]] = If[ds[[k-1]] == 0, 0, Drop[ds[[k-1]], {b}]]];
          If[k < L, ds[[k+1]] = If[ds[[k+1]] == 0, 0, Drop[ds[[k+1]], {}, {a}]]];
          Break[]
        ],
        {a, Length[chains][[k+1]]}, {b, Length[chains][[k]]}
      ]
    ],
    {k, L}
  ];
  Kom[chains, ds]
];

```

```

CFKh[L_] := Module[
  {pd = PD[L], kom = Kom[{{S[]}}, {}], inside = {}, tp = 0, pos},
  While[Length[pd] > 0,
    pos = Last[Ordering[(Length[Intersection[List @@ #, inside]]) & /@ pd]];
    kom = kom * (pd[[pos]] /. {
      X[i_, j_, k_, l_] /; (j - l == 1 || l - j > 1) =>
        Cob[Q[1] S[P[-i, j], P[k, -l]], Q[2] S[P[-i, -l], P[j, k]], 1],
      X[i_, j_, k_, l_] /; (l - j == 1 || j - l > 1) =>
        (--tp; Cob[Q[-2] S[P[-i, -j], P[k, l]], Q[-1] S[P[-i, l], P[-j, k]], 1])
    });
    (kom = Contract[kom // m[#, -#]]) & /@ Intersection[List @@ pd[[pos]], inside];
    inside = Union[inside, List @@ pd[[pos]];
    pd = Drop[pd, {pos}];
  ];
  Expand[ttp-1+Range[Length[First[kom]]].(List @@ Plus @@@ First @ kom) /.
    {Q[qp_] => qqp, S[] -> 1}]
]

```