```
<< OreGroebnerBasis.m
```

```
OreGroebnerBasis Package by Christoph Koutschan — © RISC Linz — V 0.6 (29.06.2008)
```

We read in the recurrences which annihilate the Gessel walk counting function:

```
ann = << annGessel.m;
```

Some routines for proving recurrences:

```
OperatorTimes[Q_, P_, f_[i_, j_, n_]] := Module[{a, b, c}, Factor /@
    Collect[Q /. f[i + a_., j + b_., n + c_.] :> (P /. {i → i + a, j → j + b, n → n + c}), f[__]]];
OperatorLT[P_, f_[i_, j_, n_]] := Module[{ops},
   ops = Sort[Union[Cases[{Expand[Numerator[Together[P]]]}, f[__], Infinity]]];
   Return[If[Length[ops] == 0, 0, Last[ops]]];];

(*find R,S such that P=R Q+S*)
OperatorReduce[P_, Q_, f_[i_, j_, n_]] :=
  Module[{vars, P0, lcQ, ltQ, ltP, lcP, term}, P0 = Expand[P]; R = 0; S = 0; vars = {i, j, n};
   lcQ = Coefficient[Q, ltQ = OperatorLT[Q, f[i, j, n]]];
   While[P0 =!= 0, lcP = Coefficient[P0, ltP = OperatorLT[P0, f[i, j, n]]];
    If[Min[term = (List @@ ltP - List @@ ltQ)] ≥ 0, (*ltQ|ltP*)
     term = lcP / (lcQ /. Thread[vars → vars - term]) f @@ (vars + term);
     P0 = Expand[P0 - OperatorTimes[term, Q, f[i, j, n]]];
     R += term;, P0 = Expand[P0 - lcP * ltP]; (*ltQ not|ltP*)S += lcP * ltP;];];
   Return[{R, S}];];

(*given Q,P0,find {R0,P1} such that Q P0==R0 Q+P1*)
TurnAround[Q_, P0_, f_[i_, j_, n_]] :=
  OperatorReduce[OperatorTimes[Q, P0, f[i, j, n]], Q, f[i, j, n]];

(*decide membership R in Ann f given T in
 Ann f with contant coefficients and an evaluator c*)
OperatorInAnnihilatorQ[R_, T_, f_[i_, j_, n_], c_] :=
  Module[{rn, rij, V}, If[R === 0, Return[True]];
   rn = Max[Cases[{R}, f[__, n + a_.] → a, Infinity]];
   rij =
    Max[Cases[{R}, f[i + a_., __] → a, Infinity], Cases[{R}, f[_, j + a_., _] → a, Infinity]];
   If[Union[Flatten[Table[R, {n, 0, rij}, {i, 0, n + rn}, {j, 0, n + rn}]] /. f → c] =!= {0},
    Return[False]];
   Return[OperatorInAnnihilatorQ[Last[TurnAround[T, R, f[i, j, n]]], T, f[i, j, n], c]];];
```

Define a function that computes the number of Gessel walks for concrete values:

```
c[i_Integer, j_Integer, n_Integer] :=
  Which[
   Max[i, j] > n || OddQ[n + i] || i < 0 || j < 0, 0,
   i == n, Binomial[n, j],
   j == n, KroneckerDelta[n, i],
   i == j == n == 0, 1,
   i == 0, c[0, j, n] = c[1, j, n - 1] + c[1, j + 1, n - 1],
   j == 0, c[i, 0, n] = c[i + 1, 1, n - 1] + c[i - 1, 0, n - 1] + c[i + 1, 0, n - 1],
   True,
   c[i, j, n] = c[i - 1, j, n - 1] + c[i - 1, j - 1, n - 1] + c[i + 1, j + 1, n - 1] + c[i + 1, j, n - 1]
  ];
```

The trivial recurrence which follows directly from the step set:

```
trivOp = f[i, j, n] + f[i, 1 + j, n] - f[1 + i, 1 + j, 1 + n] + f[2 + i, 1 + j, n] + f[2 + i, 2 + j, n];
```

For each of the input operators we prove that it indeed is an annihilator:

```
OperatorInAnnihilatorQ[#, trivOp, f[i, j, n], c] & /@ ann
```

```
{True, True, True, True, True, True, True, True, True, True, True, True, True, True}
```

Convert the recurrences to operator notation and compute a Groebner basis in the module that is generated by the power products of S[i] and S[j]. The option Incomplete -> True causes that the first operator that does not involve S[i] and S[j] is returned (without completing the Groebner basis computation). The computation takes about 30 hours.

```
ann = Expand[ann /. {i → 0, j → 0}] /. f[si_, sj_, n + sn_.] → S[i]^si * S[j]^sj * S[n]^sn;
```

```
gb = OreGroebnerBasis[ann, OreAlgebra[S[i], S[j], S[n]], ModuleBasis → {1, 2},
   MonomialOrder → EliminationOrder[2], Strategy → "elimination", Incomplete → True];
```

Convert the resulting operator into a recurrence:

```
rec = ApplyOreOperator[First[gb], f[n]]
```