CONJUGATE-SEMISTANDARD TABLEAU FAMILIES

This is a self-contained Haskell module for generating semistandard and conjugate semistandard tableau and tableau families. It implements Algorithm 9.5 in [1], and may also be used to verify Example 8.3.

1. Preliminaries

```haskell
module TableauFamilies where

import qualified Data.Map as M
import Data.List (delete, nub, sort, sortBy, transpose)
```

Partitions and compositions. Use *sum* to get the size of a partition.

```haskell
type Part = Int
type Partition = [Part]
type SizeOfPartition = Int

type NumberOfRows = Int
type NumberOfColumns = Int

partitionsInBox :: NumberOfRows -> NumberOfColumns -> SizeOfPartition -> [Partition]
partitionsInBox _ _ 0 = [[]]
partitionsInBox 0 _ _ = []
partitionsInBox r m n = [c : rest | c <- [1 .. m `min` n],
                                    rest <- partitionsInBox (r - 1) c (n - c)]

partitions :: SizeOfPartition -> [Partition]
partitions n = partitionsInBox n n n

conjugatePartition :: Partition -> Partition
conjugatePartition [] = []
conjugatePartition p@(a : _) = [f j | j <- [1 .. a]]
        where f j = length $ takeWhile (>= j) p

type Composition = [Part]
type SizeOfComposition = Int

compositions :: NumberOfRows -> SizeOfComposition -> [Composition]
```

Date: January 8, 2017.

```
compositions _ 0 = [[ ]]
compositions 0 _ = [ ]
compositions k n = [m : c | m ← [0 . . n], c ← compositions (k − 1) (n − m)]
```

Dominance order on compositions.

```
dominates :: Composition → Composition → Bool
p ‘dominates‘ q = and $ zipWith (⩾) (partialSums p) (partialSums q)
```

Young Diagrams.

```
type Row = Int
type Column = Int
type Box = (Row, Column)
type YoungDiagram = [Box]

youngDiagram :: Partition → YoungDiagram
youngDiagram p = [(i, j) | (i, x) ← zip [1 . .] p, j ← [1 . . x]]
```

2. Tableaux

```
type Entry = Int
type TableauRow = [Int]
type Tableau = [TableauRow]

maximumEntry :: Tableau → Int
maximumEntry [ ] = error "maximumEntry: empty tableau"
maximumEntry t = maximum $ concat [es | es ← t]
```

Mathematically a tableau is a function from the Young diagram to the set of entries. This corresponds to a Haskell map.

```
type TableauM = M.Map Box Entry

tableauToTableauM :: Tableau → TableauM
tableauToTableauM t = M.fromList $ concat $ [pairsForRow i xs | (i, xs) ← zip [1 . .] t]
    where pairsForRow i xs = [((i, j), x) | (j, x) ← zip [1 . .] xs]

tableauMToTableau :: TableauM → Tableau
tableauMToTableau tM = [row i | i ← [1 . . k]]
    where row i = [tM M.! (i, j) | j ← [1 . . lengthOfRow i]]
          lengthOfRow i = maximum [j | (i′, j) ← M.keys tM, i′ ≡ i]
          k | M.null tM = 0
            | otherwise = maximum [i | (i, _) ← M.keys tM]

changeEntry :: TableauM → Box → Entry → TableauM
```

$changeEntry\ tM\ (i, j)\ x = M.adjust\ (\backslash\_ \to x)\ (i, j)\ tM$

$insertMany :: TableauM \to [(Box, Entry)] \to TableauM$
$insertMany\ tM\ [\,] = tM$
$insertMany\ tM\ ((b, x) : rest) = insertMany\ tM'\ rest$
$\quad$ **where** $tM' = M.insert\ b\ x\ tM$

3. Total column colexicographic order on tableaux

Let Ω be totally ordered under $\leq$. Let $X = \{x_1, \ldots, x_d\}$ and $Y = \{y_1, \ldots, y_d\}$ be multisubsets of Ω, written so that $x_1 \leq \ldots \leq x_d$ and $y_1 \leq \ldots \leq y_d$. The *colexicographic order* on multisubsets of Ω is defined by $X < Y$ if and only if for some q we have $x_q < y_q$ and $x_{q+1} = y_{q+1}, \ldots, x_d = y_d$. It is a total order.

$colexGreater :: (Ord\ a) \Rightarrow [\,a\,] \to [\,a\,] \to Bool$
$colexGreater\ ys\ xs = comparePairsLex\ (>)\ \$\ zip\ (reverse\ ys)\ (reverse\ xs)$

$comparePairsLex :: (Eq\ a) \Rightarrow (a \to a \to Bool) \to [\,(a, a)\,] \to Bool$
$comparePairsLex\ \_\ [\,] = True$
$comparePairsLex\ ord\ ((x, y) : abs)$
$\quad\quad \mid x \equiv y = comparePairsLex\ ord\ abs$
$\quad\quad \mid otherwise = x\ `ord`\ y$

We define a total order on column semistandard tableau as follows: let s and t be distinct such tableaux, take the rightmost column where they differ, and compare these columns under the colexicographic order.

$columnGreater :: Tableau \to Tableau \to Bool$
$columnGreater\ t\ s = comparePairsLex\ colexGreater\ \$\ zip\ (reverse\ t')\ (reverse\ s')$
$\quad\quad$ **where** $t' = transpose\ t; s' = transpose\ s$

For use in *sortBy* convert *columnGreater* to type *Ordering* (see §10 below).

$totalOrdering :: Tableau \to Tableau \to Ordering$
$totalOrdering\ t\ s$
$\quad \mid t \equiv s = EQ$
$\quad \mid t\ `columnGreater`\ s = GT$
$\quad \mid otherwise = LT$

4. Majorization order

Let $X = \{x_1, \ldots, x_r\}$ and $Y = \{y_1, \ldots, y_r\}$ be subsets of a totally ordered set Ω, with the notation chosen so that $x_1 < x_2 < \ldots < x_r$ and $y_1 < y_2 < \ldots < y_r$. We say that Y *majorizes* X, and write $X \preceq Y$, if $x_1 < y_1$, $x_2 < y_2$, $\ldots$, $x_r < y_r$.

$$\begin{aligned}
&\mathit{majorizesList} :: (\mathit{Ord}\ a) \Rightarrow [a] \rightarrow [a] \rightarrow \mathit{Bool}\\
&\mathit{majorizesList}\ \mathit{ys}\ \mathit{xs} = \mathit{and}\ \$\ \mathit{zipWith}\ (\geqslant)\ \mathit{ys}\ \mathit{xs}
\end{aligned}$$

If s and t are conjugate-semistandard tableaux then we say that t *majorizes* s if each row of t majorizes the corresponding row of s.

$$\begin{aligned}
&\mathit{majorizes} :: \mathit{Tableau} \rightarrow \mathit{Tableau} \rightarrow \mathit{Bool}\\
&\mathit{majorizes}\ t\ s = \mathit{and}\ \$\ \mathit{zipWith}\ \mathit{majorizesList}\ t\ s\\
&\\
&\mathit{incomparable}\ s\ t = \neg\ (\mathit{majorizes}\ s\ t) \wedge \neg\ (\mathit{majorizes}\ t\ s)
\end{aligned}$$

Neighbours in the majorization order. The neighbours of a conjugate-semistandard tableau are obtained by considering each position in turn, and decrementing the entry in this position when this gives a conjugate-semistandard tableau.

$$\begin{aligned}
&\textbf{type}\ \mathit{ConjugateSemistandardTableau} = \mathit{Tableau}\\
&\\
&\mathit{downNeighbours} :: \mathit{ConjugateSemistandardTableau} \rightarrow [\mathit{ConjugateSemistandardTableau}]\\
&\mathit{downNeighbours}\ t =\\
&\qquad [\mathit{tableauMToTableau}\ \mathit{tM} \mid \mathit{tM} \leftarrow \mathit{downNeighboursM}\ \$\ \mathit{tableauToTableauM}\ t]\\
&\\
&\mathit{downNeighboursM} :: \mathit{TableauM} \rightarrow [\mathit{TableauM}]\\
&\mathit{downNeighboursM}\ \mathit{tM} = [\mathit{tM}' \mid (i,j) \leftarrow \mathit{M.keys}\ \mathit{tM},\\
&\qquad\qquad\qquad\qquad\qquad\qquad \mathit{Just}\ \mathit{tM}' \leftarrow [\mathit{decrement}\ \mathit{tM}\ (i,j)]]\\
&\\
&\mathit{decrement} :: \mathit{TableauM} \rightarrow \mathit{Box} \rightarrow \mathit{Maybe}\ \mathit{TableauM}\\
&\mathit{decrement}\ \mathit{tM}\ (i,j) \mid e \not\equiv 1 \wedge \mathit{rowCheck} \wedge \mathit{columnCheck} = \mathit{Just}\ \mathit{tM}'\\
&\qquad\qquad\qquad\qquad \mid \mathit{otherwise} = \mathit{Nothing}\\
&\quad \textbf{where}\ e = \mathit{tMM}\,.\,!\,(i,j)\\
&\qquad\qquad \mathit{rowCheck} = j \equiv 1 \vee e > \mathit{tMM}\,.\,!\,(i,j-1) + 1\\
&\qquad\qquad \mathit{columnCheck} = i \equiv 1 \vee e > \mathit{tMM}\,.\,!\,(i-1,j)\\
&\qquad\qquad \mathit{tM}' = \mathit{changeEntry}\ \mathit{tM}\ (i,j)\ (e-1)
\end{aligned}$$

Closed families. A *closed conjugate-semistandard tableaux family* is a set of conjugate-semistandard tableau downwardly closed under the majorization order. We say such sets are *downsets*; any downset is determined by its maximal elements.

Take a list $\mathcal{T}$ of conjugate-semistandard tableaux, with head t. We take t if it is maximal in $\mathcal{T}$ and, in either case, continue with the tableaux not majorized by t. This ensures that if s is not maximal, or appears multiple times in $\mathcal{T}$, then either s is thrown out because it is majorized by some earlier tableau, or it thrown out when it reaches the head of the list, and is majorized by some later tableau.

$$\textbf{type}\ \mathit{Maximal} = \mathit{ConjugateSemistandardTableau}$$

$$
\begin{aligned}
&\mathit{maximals} :: [\mathit{ConjugateSemistandardTableau}] \to [\mathit{Maximal}]\\
&\mathit{maximals}\ [\,] = [\,]\\
&\mathit{maximals}\ (x : \mathit{xs}) \mid \mathit{xMaximal} = x : \mathit{maximals}\ \mathit{xs}'\\
&\qquad\qquad\qquad\quad \mid \mathit{otherwise} = \mathit{maximals}\ \mathit{xs}'\\
&\qquad \mathbf{where}\ \mathit{xMaximal} = \mathit{and}\ [\neg\ (y\ \text{`}\mathit{majorizes}\text{`}\ x) \mid y \leftarrow \mathit{xs}, y \not\equiv x]\\
&\qquad\qquad\quad \mathit{xs}' = [y \mid y \leftarrow \mathit{xs}, \neg\ (x\ \text{`}\mathit{majorizes}\text{`}\ y)]
\end{aligned}
$$

Given a list of incomparable conjugate-semistandard tableaux $\mathcal{M}$, the downset having these tableaux as its maximal elements may be constructed as follows:

(1) Put all tableaux in $\mathcal{M}$ in the family;
(2) Let $\mathcal{T}$ be the list of tableaux one step below a maximal $s \in \mathcal{M}$, in the majorization order. Repeat (1) with the maximal elements of $\mathcal{T}$.

Note that if tableaux t, $u \in \mathcal{T}$ satisfy $t \succeq u$ then we will see u in the downset on t, so it is safe to discard u. Indeed, this ensures that no conjugate-semistandard tableau can appear twice, because whenever we put tableaux into the family, they are all incomparable, and (except in the first step) each is majorized by a tableau already in the family

$$
\begin{aligned}
&\mathbf{type}\ \mathit{TableauFamily} = [\mathit{ConjugateSemistandardTableau}]\\
\\
&\mathit{downSetSorted} :: [\mathit{Maximal}] \to \mathit{TableauFamily}\\
&\mathit{downSetSorted}\ \mathit{ss} = \mathit{sortBy}\ \mathit{totalOrdering}\ (\mathit{downSetOnMaximals}\ \mathit{ss})\\
\\
&\mathit{downSetOnMaximals} :: [\mathit{Maximal}] \to \mathit{TableauFamily}\\
&\mathit{downSetOnMaximals}\ [\,] = [\,]\\
&\mathit{downSetOnMaximals}\ \mathit{ss} = \mathit{ss} +\!\!+\ \mathit{downSetOnMaximals}\ \mathit{ss}''\\
&\qquad\quad \mathbf{where}\ \mathit{ss}' = \mathit{concat}\ [\mathit{downNeighbours}\ s \mid s \leftarrow \mathit{ss}]\\
&\qquad\qquad\qquad \mathit{ss}'' = \mathit{maximals}\ \mathit{ss}'\\
\\
&\mathit{downSet} :: \mathit{ConjugateSemistandardTableau} \to \mathit{TableauFamily}\\
&\mathit{downSet}\ t = \mathit{downSetOnMaximals}\ [t]
\end{aligned}
$$

This gives a convenient way to generate all conjugate-semistandard tableau of a given shape that is not much slower than more sophisticated methods using iterated Pieri's rule removal of boxes (see §9.2 below).

$$
\begin{aligned}
&\mathbf{type}\ \mathit{MaximumPermittedEntry} = \mathit{Int}\\
\\
&\mathit{conjugateSemistandardTableauxByMaj} :: \mathit{Partition} \to \mathit{MaximumPermittedEntry}\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \to [\mathit{ConjugateSemistandardTableau}]\\
&\mathit{conjugateSemistandardTableauxByMaj}\ [\,]\ \_ = [[\,]]\\
&\mathit{conjugateSemistandardTableauxByMaj}\ p@(a : \_)\ k
\end{aligned}
$$

$\quad\quad \mid a > k = [\,]$
$\quad\quad \mid \mathit{otherwise} = \mathit{downSet}\ [[k - a + 1 \mathinner{..} k] \mid a \leftarrow p]$

$\mathit{totalOrderCSSYTs} :: \mathit{Partition} \rightarrow \mathit{MaximumPermittedEntry}$
$\quad\quad\quad\quad\quad\quad\quad\quad \rightarrow [\mathit{ConjugateSemistandardTableau}]$
$\mathit{totalOrderCSSYTs}\ p\ m = \mathit{sortBy}\ \mathit{totalOrdering}\ (\mathit{conjugateSemistandardTableaux}\ p\ m)$

$\mathit{numberOfCSSYTs} :: \mathit{Partition} \rightarrow \mathit{MaximumPermittedEntry} \rightarrow \mathit{Int}$
$\mathit{numberOfCSSYTs}\ p\ m = \mathit{length}\ \$\ \mathit{conjugateSemistandardTableaux}\ p\ m$

For example, $\mathit{printTableaux}\ \$\ \mathit{totalOrderCSSYTs}\ [2, 2]\ 4$ evaluates to

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 1 & 2 \\ \hline\end{array}\,,\quad \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 1 & 3 \\ \hline\end{array}\,,\quad \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 2 & 3 \\ \hline\end{array}\,,\quad \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 & 3 \\ \hline\end{array}\,,\quad \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 & 4 \\ \hline\end{array}\,,\quad \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 & 4 \\ \hline\end{array}\,.$$

5. Constructing conjugate-semistandard tableau families

We use a refinement of the algorithm used to generate the downset on a conjugate-semistandard tableau. Start with a list of candidate maximal tableaux $\mathcal{M}$ and the empty family. If s is at the head of $\mathcal{M}$ then *either*

- declare that s is not in the family, *or*
- insert the downset on s into the family and remove all candidate maximal tableau t from $\mathcal{M}$ that are comparable to s.

Then repeat with the tail of $\mathcal{M}$.

$\mathbf{type}\ N = \mathit{Int}$
$\mathbf{type}\ \mathit{SizeOfFamily} = \mathit{Int}$
$\mathbf{type}\ \mathit{CandidateMaximal} = \mathit{ConjugateSemistandardTableau}$

$\mathit{tableauFamiliesMS} :: N \rightarrow ([\mathit{CandidateMaximal}], [\mathit{Maximal}], \mathit{SizeOfFamily})$
$\quad\quad\quad\quad\quad\quad\quad\quad \rightarrow [[\mathit{Maximal}]]$
$\mathit{tableauFamiliesMS}\ n\ ([\,], \mathit{ts}, l)$
$\quad\quad \mid l \equiv n = [\mathit{ts}]$
$\quad\quad \mid \mathit{otherwise} = [\,]$
$\mathit{tableauFamiliesMS}\ n\ ((s : \mathit{ms}), \mathit{ts}, l)$
$\quad\quad \mid l > n = [\,]$
$\quad\quad \mid l \equiv n = [\mathit{ts}]$
$\quad\quad \mid \mathit{otherwise} = \mathit{tableauFamiliesMS}\ n\ (\mathit{ms}', \mathit{ts}', l') +\!\!+\ \mathit{tableauFamiliesMS}\ n\ (\mathit{ms}, \mathit{ts}, l)$
$\quad\quad\quad\quad \mathbf{where}\ \mathit{ms}' = [s' \mid s' \leftarrow \mathit{ms}, s\ \text{`}\mathit{incomparable}\text{`}\ s']$
$\quad\quad\quad\quad\quad\quad\quad\quad \mathit{ts}' = s : \mathit{ts}$
$\quad\quad\quad\quad\quad\quad\quad\quad l' = l + \mathit{length}\ [u \mid u \leftarrow \mathit{downSet}\ s, \mathit{and}\ [u\ \text{`}\mathit{incomparable}\text{`}\ t \mid t \leftarrow \mathit{ts}]]$

$\mathit{tableauFamiliesM} :: \mathit{Partition} \rightarrow N \rightarrow \mathit{MaximumPermittedEntry} \rightarrow [[\mathit{Maximal}]]$

$$\begin{aligned}
&\mathit{tableauFamiliesM}\ p\ n\ k = \mathit{tableauFamiliesMS}\ n\ (\mathit{ms}, [\,], 0)\\
&\qquad\quad \mathbf{where}\ \mathit{ms} = \mathit{conjugateSemistandardTableaux}\ p\ k
\end{aligned}$$

$$\begin{aligned}
&\mathit{tableauFamilies} :: \mathit{Partition} \to N \to \mathit{MaximumPermittedEntry} \to [\,\mathit{TableauFamily}\,]\\
&\mathit{tableauFamilies}\ p\ n\ k = [\,\mathit{downSetSorted}\ \mathit{ss} \mid \mathit{ss} \leftarrow \mathit{tableauFamiliesM}\ p\ n\ k\,]
\end{aligned}$$

6. Weights and types of tableau and tableau families

$$\begin{aligned}
&\mathbf{type}\ \mathit{Weight} = [\,\mathit{Multiplicity}\,]\\
&\mathbf{type}\ \mathit{Multiplicity} = \mathit{Int}
\end{aligned}$$

$$\begin{aligned}
&\mathit{weightT} :: \mathit{Tableau} \to [\,\mathit{Multiplicity}\,]\\
&\mathit{weightT}\ t = [\,\mathit{numberOf}\ x \mid x \leftarrow [1\,..\,\mathit{maximumEntry}\ t\,]\,]\\
&\qquad\quad \mathbf{where}\ \mathit{numberOf}\ x = \mathit{sum}\ [\,\mathit{countR}\ x\ r \mid r \leftarrow t\,]\\
&\qquad\qquad\qquad\ \ \mathit{countR}\ x\ r = \mathit{length}\ [\,y \mid y \leftarrow r, y \equiv x\,]
\end{aligned}$$

$$\begin{aligned}
&\mathit{weight} :: [\,\mathit{ConjugateSemistandardTableau}\,] \to \mathit{Weight}\\
&\mathit{weight}\ \mathit{ts} = \mathit{sumWeights}\ [\,\mathit{weightT}\ t \mid t \leftarrow \mathit{ts}\,]
\end{aligned}$$

$$\begin{aligned}
&\mathit{weightM} :: [\,\mathit{Maximal}\,] \to \mathit{Weight}\\
&\mathit{weightM}\ \mathit{ss} = \mathit{weight}\ \$\ \mathit{downSetOnMaximals}\ \mathit{ss}
\end{aligned}$$

$$\begin{aligned}
&\mathit{addWeights} :: \mathit{Weight} \to \mathit{Weight} \to \mathit{Weight}\\
&\mathit{addWeights}\ u\ v \mid \mathit{length}\ u < \mathit{length}\ v = \mathit{addWeights}\ v\ u\\
&\qquad\qquad\qquad\quad \mid \mathit{otherwise} = \mathit{zipWith}\ (+)\ u\ v +\!\!+\ \mathit{drop}\ (\mathit{length}\ v)\ u
\end{aligned}$$

$$\begin{aligned}
&\mathit{sumWeights} :: [\,\mathit{Weight}\,] \to \mathit{Weight}\\
&\mathit{sumWeights}\ \mathit{ps} = \mathit{foldr1}\ \mathit{addWeights}\ \mathit{ps}
\end{aligned}$$

$$\mathbf{type}\ \mathit{PType} = \mathit{Partition}$$

$$\begin{aligned}
&\mathit{ptype} :: [\,\mathit{ConjugateSemistandardTableau}\,] \to \mathit{Partition}\\
&\mathit{ptype}\ \mathit{ts} = \mathit{conjugatePartition}\ \$\ \mathit{weight}\ \mathit{ts}
\end{aligned}$$

$$\begin{aligned}
&\mathit{ptypeM} :: [\,\mathit{Maximal}\,] \to \mathit{Partition}\\
&\mathit{ptypeM}\ \mathit{ss} = \mathit{ptype}\ \$\ \mathit{downSetOnMaximals}\ \mathit{ss}
\end{aligned}$$

7. Tableau families of maximal weight (equivalently, minimal type)

A tableau family of maximal weight (equivalently minimal type) is closed. To select the closed families of maximal weight we use a similar trick to *maximals* to throw out families of non-maximal weight, with a small change because there may be several different families with the same maximal weight

```
type Mu = Partition
type Nu = Partition

closedWeightsM :: Mu → N → MaximumPermittedEntry → [(Weight, [Maximal])]
closedWeightsM p n k = sort [(weightM ss, ss) | ss ← tableauFamiliesM p n k]

maximalWeightsM :: Mu → N → MaximumPermittedEntry → [(Weight, [Maximal])]
maximalWeightsM p n k = takeMaximalWeights $ closedWeightsM p n k

maximalWeights :: Mu → N → MaximumPermittedEntry → [Weight]
maximalWeights p n k = [w | (w, _) ← maximalWeightsM p n k]

takeMaximalWeights :: [(Weight, [Maximal])] → [(Weight, [Maximal])]
takeMaximalWeights [] = []
takeMaximalWeights ((u, ss) : uss)
        | pMaximal = (u, ss) : takeMaximalWeights uss'
            | otherwise = takeMaximalWeights uss'
        where pMaximal = and [¬ (v `dominates` u) | (v, _) ← uss, v ≢ u]
                   uss' = [(v, ts) | (v, ts) ← uss, u ≡ v ∨ ¬ (u `dominates` v)]

minimalTypes :: Mu → N → MaximumPermittedEntry → [PType]
minimalTypes p n k = sort [conjugatePartition q | q ← maximalWeights p n k]
```

The greatest entry in a conjugate-semistandard tableau family of shape μ^n is $m+n-1$.

```
minimalTypesA :: Mu → N → [PType]
minimalTypesA p n = minimalTypes p n (sum p + n − 1)

maximalWeightsA :: Mu → N → [PType]
maximalWeightsA p n = maximalWeights p n (sum p + n − 1)

minMaxs :: Mu → N → [PType]
minMaxs p n = minimalTypesA p n `meet` maximalWeightsA (conjugatePartition p) n
```

Case $\mu = (3)$. Identify conjugate-semistandard (3)-tableau with 3-subsets of $\mathbf{N}$. The downset on $\{a, b\}$ where $a < b$ consists of all $\{1, 2\}, \ldots, \{1, a\}, \ldots, \{a, a+1\}, \ldots, \{a, b\}$. The sets with common least element m contain $2(b - m)$ elements, so the type of the downset is a partition of $\sum_{m=1}^{a} 2(b - m) = a(2b - a - 1)$. It follows that the downset on $\{r, a, b\}$ contains $(a - r)(2b - a - r - 1)$ sets with least element r. Therefore $\{r, a, b\}$ is a candidate maximal in a set family of size n only if $(a - r)(2b - a - r - 1) < 3n$.

$$\begin{aligned}
&\mathit{threeFamiliesCandidateMaximals}\ n = \mathit{ms}' \\
&\qquad\qquad \textbf{where}\ \mathit{ms} = \mathit{conjugateSemistandardTableaux}\ [3]\ (n+2) \\
&\qquad\qquad\qquad\quad \mathit{ms}' = [t \mid t@[[r,a,b]] \leftarrow \mathit{ms}, (a-r)*(2*b-a-r-1) \leqslant 3*n]
\end{aligned}$$

$$\begin{aligned}
&\mathit{threeFamilies}\ n = \\
&\qquad \mathit{tableauFamiliesMS}\ n\ (\mathit{threeFamiliesCandidateMaximals}\ n, [\,], 0)
\end{aligned}$$

$$\mathit{threeTypes}\ n = \mathit{collectSorted}\ \$\ \mathit{sort}\ \$\ [\mathit{ptypeM}\ \mathit{ms} \mid \mathit{ms} \leftarrow \mathit{threeFamilies}\ n]$$

$$\mathit{threeTypesMultiple}\ n = [(p,m) \mid (p,m) \leftarrow \mathit{threeTypes}\ n, m \geqslant 2]$$

$$\begin{aligned}
&\mathit{threeTypesClosedNonMinimal}\ n = [(\mathit{conjugatePartition}\ w, \mathit{ms}) \mid (w, \mathit{ms}) \leftarrow \mathit{vs}\ \text{‘}\mathit{diff}\text{‘}\ \mathit{vs}'] \\
&\qquad\qquad \textbf{where}\ \mathit{vs} = [(\mathit{weightM}\ \mathit{ms}, \mathit{ms}) \mid \mathit{ms} \leftarrow \mathit{threeFamilies}\ n] \\
&\qquad\qquad\qquad\quad \mathit{vs}' = \mathit{takeMaximalWeights}\ \mathit{vs}
\end{aligned}$$

7.1. **Closed non-maximal families.** It is an open question whether every closed conjugate-semistandard tableau family corresponds to a summand of a generalized Foulkes module.

$$\begin{aligned}
&\mathit{closedNonMaximalWeightsM} :: \mathit{Mu} \to N \to \mathit{MaximumPermittedEntry} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \to [(\mathit{Weight}, [\mathit{Maximal}])] \\
&\mathit{closedNonMaximalWeightsM}\ p\ n\ k \\
&\qquad = \mathit{closedWeightsM}\ p\ n\ k\ \text{‘}\mathit{diff}\text{‘}\ \mathit{maximalWeightsM}\ p\ n\ k
\end{aligned}$$

$$\begin{aligned}
&\mathit{closedNonMaximalWeights} :: \mathit{Mu} \to N \to \mathit{MaximumPermittedEntry} \to [\mathit{Weight}] \\
&\mathit{closedNonMaximalWeights}\ p\ n\ k \\
&\qquad = [w \mid (w, _) \leftarrow \mathit{closedNonMaximalWeightsM}\ p\ n\ k]
\end{aligned}$$

7.2. **Unique families.** Corollary 9.10 in [1] characterizes the partitions μ and $n \in \mathbf{N}$ such that there is a unique conjugate-semistandard tableau family of shape μ^n.

$$\begin{aligned}
&\mathit{uniqueFamily}\ p@(a : _)\ n = l \equiv 1 \\
&\qquad\qquad \textbf{where}\ l = \mathit{length}\ \$\ \mathit{maximalWeights}\ p\ n\ (n + a - 1)
\end{aligned}$$

8. Example 8.3 in [1]

Define

$$u = \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 4 & \multicolumn{1}{c}{} \\ \cline{1-1}\end{array},\ v = \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 2 & \multicolumn{1}{c}{} \\ \cline{1-1}\end{array},\ w = \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 3 & \multicolumn{1}{c}{} \\ \cline{1-1}\end{array},\ x = \begin{array}{|c|c|}\hline 1 & 4 \\ \hline 2 & \multicolumn{1}{c}{} \\ \cline{1-1}\end{array}.$$

These tableaux are incomparable in the majorization order.

$$u = [[1,2],[4]]; v = [[2,3],[2]]; w = [[1,3],[3]]; x = [[1,4],[2]]$$

The tableaux majorized by one of u, v, w, x are constructed below.

$$\begin{aligned}
&\mathit{ts} = \mathit{sortBy}\ \mathit{totalOrdering}\ \$\ \mathit{downSetOnMaximals}\ [u, v, w, x]\\
&[\mathit{t1}, \mathit{t2}, \mathit{t3}, \mathit{t4}, \mathit{t5}, \mathit{t6}, \mathit{t7}, \mathit{t8}, \mathit{t9}, \mathit{t10}] = \mathit{ts}\\
&\mathit{checkLabels} = (u \equiv \mathit{t4}) \wedge (v \equiv \mathit{t7}) \wedge (w \equiv \mathit{t8}) \wedge (x \equiv \mathit{t10})
\end{aligned}$$

It is convenient to have these tableaux printed in this notation.

$$\begin{aligned}
&\mathit{showExT} :: \mathit{ConjugateSemistandardTableau} \to \mathit{String}\\
&\mathit{showExT}\ s \mid s \equiv u = \texttt{"u"}\\
&\qquad\qquad \mid s \equiv v = \texttt{"v"}\\
&\qquad\qquad \mid s \equiv w = \texttt{"w"}\\
&\qquad\qquad \mid s \equiv x = \texttt{"x"}\\
&\qquad\qquad \mid s \in \mathit{ts} = \texttt{"t"} +\!\!+\ \mathit{show}\ (\mathit{position}\ s\ \mathit{ts} + 1)\\
&\qquad\qquad \mid \mathit{otherwise} = \mathit{error}\ \$\ \texttt{"showExT: "} +\!\!+\ \mathit{show}\ s
\end{aligned}$$

The conjugate-semistandard tableau families and conjugte-semistandard tableau family tuples defined in Example 8.3 are as follows.

$$\begin{aligned}
&\mathit{sm} = \mathit{ts}\ `\mathit{diff}`\ [u, v, w, x]\\
&\mathit{add}\ \mathit{ss}\ \mathit{ys} = \mathit{sortBy}\ \mathit{totalOrdering}\ \$\ \mathit{ss} +\!\!+\ \mathit{ys}\\
&\mathit{ss1} = \mathit{sm}\ `\mathit{add}`\ [u, v]; \mathit{ss2} = \mathit{sm}\ `\mathit{add}`\ [w, x]; \mathit{ss3} = \mathit{sm}\ `\mathit{add}`\ [u, x]\\
&\mathit{ss4} = \mathit{sm}\ `\mathit{add}`\ [v, w]; \mathit{ss5} = \mathit{sm}\ `\mathit{add}`\ [u, w]; \mathit{ss6} = \mathit{sm}\ `\mathit{add}`\ [v, x]\\
&\mathit{tft1} = [\mathit{ss1}, \mathit{ss2}]; \mathit{tft2} = [\mathit{ss3}, \mathit{ss4}]; \mathit{tft3} = [\mathit{ss5}, \mathit{ss6}]; \mathit{tft4} = [\mathit{ss1}, \mathit{ss5}]; \mathit{tft5} = [\mathit{ss6}, \mathit{ss2}]
\end{aligned}$$

We claim that these are all closed conjugate-semistandard tableau family tuples of shape $(2,2)^{(8,8)}$ and type $(4^4, 3^5, 2^5, 1^7)$.

$$\mathit{exampleWeight} = \mathit{conjugatePartition}\ [4,4,4,4,3,3,3,3,3,2,2,2,2,2,1,1,1,1,1,1,1]$$

$$\mathit{closedWeightsM88} = \mathit{closedWeightsM}\ [2,1]\ 8\ 4$$

$$\begin{aligned}
\mathit{exampleTuplesM} = [((p, \mathit{ms}), (p', \mathit{ms}')) \mid\ & (p, \mathit{ms}) \leftarrow \mathit{closedWeightsM88},\\
& (p', \mathit{ms}') \leftarrow \mathit{closedWeightsM88},\\
& p \leqslant p',\\
& p\ `\mathit{addWeights}`\ p' \equiv \mathit{exampleWeight}]
\end{aligned}$$

$$\begin{aligned}
&\mathit{exampleTuplesF}\\
&\quad = [(\mathit{downSetSorted}\ \mathit{ms}, \mathit{downSetSorted}\ \mathit{ms}')\\
&\qquad\quad \mid ((_, \mathit{ms}), (_, \mathit{ms}')) \leftarrow \mathit{exampleTuplesM}]
\end{aligned}$$

$$\begin{aligned}
&\mathit{exampleTuplesFT}\\
&\quad = [(\mathit{identifyFamily}\ \mathit{ts}, \mathit{identifyFamily}\ \mathit{ss}, \mathit{ts}, \mathit{ss}) \mid (\mathit{ts}, \mathit{ss}) \leftarrow \mathit{exampleTuplesF}]
\end{aligned}$$

$$\mathit{exampleTupleLabels} = [(i, j) \mid (\mathit{Just}\ i, \mathit{Just}\ j, \_, \_) \leftarrow \mathit{exampleTuplesFT}]$$

```
identifyFamily ss | ss ∈ sss = Just $ 1 + position ss sss
                  | otherwise = Nothing
        where sss = [ss1, ss2, ss3, ss4, ss5, ss6]
```

exampleTupleLabels evaluates to `[(4,3),(6,5),(6,2),(1,5),(1,2)]`. Thus the first tableau family tuple found by Haskell is `(ss4, ss3)`, which is up to the order of the two families, the same as `tft2` above, and so on.

9. Tableau families of lexicographically minimal type

In this section we implement Algorithm 9.5 in [1].

9.1. Entry order on conjugate-semistandard tableau. A further order will be useful: we first compare the multisets of entries colexicographically, then use the total column colexicographic order to break ties.

```
entryGreater :: Tableau → Tableau → Bool
entryGreater t s | xs ≡ ys = columnGreater t s
                 | otherwise = colexGreater ys xs
        where xs = sort (concat s)
              ys = sort (concat t)

entryOrdering :: Tableau → Tableau → Ordering
entryOrdering t s
    | t ≡ s = EQ
    | t `entryGreater` s = GT
    | otherwise = LT

entryOrderCSSYTs :: Partition → MaximumPermittedEntry
                            → [ConjugateSemistandardTableau]
entryOrderCSSYTs p k
    = sortBy entryOrdering (conjugateSemistandardTableaux p k)
```

9.2. Young and Pieri removal of boxes.

```
type NumberOfBoxesToRemove = Int
type PartitionChain = [Partition]
type ReversedComposition = Composition

youngRemove :: NumberOfBoxesToRemove → Partition → [Partition]
youngRemove 0 p = [p]
youngRemove r [] = []
youngRemove r [x] | x > r = [[x − r]]
                  | r ≡ x = [[]]
```

$$\mid \mathit{otherwise} = [\,]$$

$$\mathit{youngRemove}\ r\ (x : y : \mathit{zs})$$
$$= [(x - r') : p \mid r' \leftarrow [0\,..\,(x - y)\ \text{`}\mathit{min}\text{`}\ r], p \leftarrow \mathit{youngRemove}\ (r - r')\ (y : \mathit{zs})]$$

$$\mathit{pieriRemove} :: \mathit{NumberOfBoxesToRemove} \rightarrow \mathit{Partition} \rightarrow \mathit{PartitionChain}$$
$$\mathit{pieriRemove}\ r\ p = [\mathit{conjugatePartition}\ q \mid q \leftarrow \mathit{youngRemove}\ r\ \$\ \mathit{conjugatePartition}\ p]$$

$$\mathit{pieriRemoveMany} :: \mathit{ReversedComposition} \rightarrow \mathit{Partition} \rightarrow [\mathit{PartitionChain}]$$
$$\mathit{pieriRemoveMany}\ [\,]\ p = [[p]]$$
$$\mathit{pieriRemoveMany}\ (c : \mathit{cs})\ p =$$
$$[p : \mathit{qs} \mid q \leftarrow \mathit{pieriRemove}\ c\ p, \mathit{qs} \leftarrow \mathit{pieriRemoveMany}\ \mathit{cs}\ q]$$

To construct tableaux it is most useful to have the boxes removed at each step.

$$\textbf{type}\ \mathit{BoxChain} = [[\mathit{Box}]]$$

$$\mathit{partitionChainToBoxChain} :: \mathit{PartitionChain} \rightarrow \mathit{BoxChain}$$
$$\mathit{partitionChainToBoxChain}\ \mathit{qs} = \mathit{differences}\ [\mathit{youngDiagram}\ q \mid q \leftarrow \mathit{qs}]$$

$$\mathit{differences} :: [\mathit{YoungDiagram}] \rightarrow [[\mathit{Box}]]$$
$$\mathit{differences}\ [\,] = [\,]$$
$$\mathit{differences}\ [d] = [\,]$$
$$\mathit{differences}\ (d : d' : \mathit{es}) = d\ \text{`}\mathit{diff}\text{`}\ d' : \mathit{differences}\ (d' : \mathit{es})$$

Conjugate semistandard tableau of given weight. Pieri removal gives a faster way to generate all conjugate semistandard tableaux then the method seen in §4. Note that the weight is reversed in the second function below: boxes removed first get the greatest number.

$$\mathit{cssytsWithWeight} :: \mathit{Weight} \rightarrow \mathit{Partition} \rightarrow [\mathit{ConjugateSemistandardTableau}]$$
$$\mathit{cssytsWithWeight}\ w\ p =$$
$$[\mathit{partitionChainToTableau}\ \mathit{bss} \mid \mathit{bss} \leftarrow \mathit{pieriRemoveMany}\ (\mathit{reverse}\ w)\ p]$$

$$\mathit{boxChainToTableauM} :: \mathit{BoxChain} \rightarrow \mathit{TableauM}$$
$$\mathit{boxChainToTableauM}\ \mathit{bss} = \mathit{insertMany}\ (M.\mathit{empty})\ \mathit{bxs}$$
$$\textbf{where}\ \mathit{bxs} = \mathit{concat}\ [[(b, k) \mid b \leftarrow \mathit{bs}] \mid (\mathit{bs}, k) \leftarrow \mathit{zip}\ \mathit{bss}\ (\mathit{reverse}\ [1\,..\,k])]$$
$$k = \mathit{length}\ \mathit{bss}$$

$$\mathit{partitionChainToTableau} :: \mathit{PartitionChain} \rightarrow \mathit{Tableau}$$
$$\mathit{partitionChainToTableau} = \mathit{tableauMToTableau} \circ \mathit{boxChainToTableauM}$$
$$\circ\ \mathit{partitionChainToBoxChain}$$

$$\begin{aligned}&\mathit{conjugateSemistandardTableaux} :: \mathit{Partition} \to \mathit{MaximumPermittedEntry}\\&\qquad\qquad\qquad\qquad\qquad\qquad \to [\,\mathit{ConjugateSemistandardTableau}\,]\\&\mathit{conjugateSemistandardTableaux}\ p\ k\\&\qquad = \mathit{concat}\ [\,\mathit{cssytsWithWeight}\ w\ p \mid w \leftarrow \mathit{compositions}\ k\ (\mathit{sum}\ p)]\end{aligned}$$

9.3. k **statistic.** At step j we have $(k_1, \ldots, k_{j-1}) = (\ell_1^{c_1}, \ldots, \ell_s^{c_s})$ and the *target* is to find d more conjugate-semistandard tableaux. We choose the maximum k such that

$$\sum |\mathrm{CSSYT}(\vartheta, k)| \le d$$

where the the sum is over all chains

$$\mu \to_{c_1} \vartheta_1 \to_{c_2} \cdots \to_{c_{\ell-1}} \vartheta_{\ell-1} \to_{c_\ell} \vartheta$$

ending in the partition ϑ; here the notation indicates that we perform a Pieri removal of c_1 boxes from μ, then c_2 boxes from the resulting partition ϑ_1, and so on. (The first step when $j = 1$ is distinguished in the description of Algorithm 9.5 in [1], but simply corresponds to the case when the only chain considered is the trivial one, ending in μ.)

$$\begin{aligned}&\mathit{chainsWithSizes} :: \mathit{Mu} \to [\,\mathit{NumberOfBoxesToRemove}\,] \to \mathit{MaximumPermittedEntry}\\&\qquad\qquad\qquad \to [\,(\mathit{PartitionChain}, \mathit{Int})]\\&\mathit{chainsWithSizes}\ p\ cs\ k =\\&\qquad [\,(\mathit{qChain}, \mathit{numberOfCSSYTs}\ (\mathit{last}\ \mathit{qChain})\ k) \mid \mathit{qChain} \leftarrow \mathit{pieriRemoveMany}\ cs\ p\,]\end{aligned}$$

$$\begin{aligned}&\mathit{chainSize} :: \mathit{Mu} \to [\,\mathit{NumberOfBoxesToRemove}\,] \to \mathit{MaximumPermittedEntry} \to \mathit{Int}\\&\mathit{chainSize}\ p\ cs\ k = \mathit{sum}\ [\,t \mid (\_, t) \leftarrow \mathit{chainsWithSizes}\ p\ cs\ k\,]\end{aligned}$$

For example, *chainsWithSizes* $[2,2]$ $[\,]$ 3 evaluates to $[([[2,2]],6)]$, corresponding to the 6 conjugate-semistandard $(2,2)$-tableaux with maximum entry 3 (these can be produced using *printTableaux* \$ *totalOrderCSSYTs* $[2,2]$ 3), and *chainsWithSizes* $[2,2]$ $[1,1]$ 2 evaluates to $[([[2,2],[2,1],[1,1]],3),([[2,2],[2,1],[2]],1)]$, corresponding to the tableaux of the two forms

$$\begin{array}{|c|c|}\hline \star & x\\\hline \star & y\\\hline\end{array}\qquad \begin{array}{|c|c|}\hline \star & \star\\\hline x & y\\\hline\end{array},$$

where $x < y$ and $\star$ denotes an unspecified entry not exceeding 2. There are 3 tableaux of the first form, and a unique tableau of the second.

$$\begin{aligned}&\mathbf{type}\ K = \mathit{Int}\\&\mathbf{type}\ \mathit{NumberOfNewTableaux} = \mathit{Int}\\&\mathbf{type}\ \mathit{Target} = \mathit{Int}\end{aligned}$$

$$\begin{aligned}&\mathit{kPair} :: \mathit{Mu} \to [\,\mathit{NumberOfBoxesToRemove}\,] \to \mathit{Target}\\&\qquad\quad \to \mathit{Maybe}\ (K, \mathit{NumberOfNewTableaux})\\&\mathit{kPair}\ p\ cs\ \mathit{target} = \mathit{maybeLast}\ \$\ \mathit{takeWhile}\ (\lambda(\_, b) \to b \leqslant \mathit{target})\\&\qquad\qquad\qquad\qquad\qquad\qquad [\,(k, \mathit{chainSize}\ p\ cs\ k) \mid k \leftarrow [0\,..]]\end{aligned}$$

9.4. **Chains to tableau families.** Each box removed in the step $\vartheta_{r-1} \to_{c_\ell} \vartheta_r$ is filled with $\ell_r + 1$. For example *partitionChainToFamily* $[[4,2],[3,1],[2,1]]$ 3 $[5,3]$ evaluates to

$$\begin{array}{|c|c|c|c|}\hline 1&2&4&6\\\hline 1&6\\\cline{1-2}\end{array}\,,\ \begin{array}{|c|c|c|c|}\hline 1&2&4&6\\\hline 2&6\\\cline{1-2}\end{array}\,,\ \begin{array}{|c|c|c|c|}\hline 1&3&4&6\\\hline 1&6\\\cline{1-2}\end{array}\,,\ \begin{array}{|c|c|c|c|}\hline 1&2&4&6\\\hline 3&6\\\cline{1-2}\end{array}\,,\ \begin{array}{|c|c|c|c|}\hline 1&3&4&6\\\hline 2&6\\\cline{1-2}\end{array}\,,\ \begin{array}{|c|c|c|c|}\hline 2&3&4&6\\\hline 2&6\\\cline{1-2}\end{array}\,,\ \begin{array}{|c|c|c|c|}\hline 1&3&4&6\\\hline 1&6\\\cline{1-2}\end{array}\,,\ \begin{array}{|c|c|c|c|}\hline 2&3&4&6\\\hline 3&6\\\cline{1-2}\end{array}\,.$$

with $5+1=6$ placed in the boxes of $(4,2)/(3,1)$, $3+1$ placed in the unique box of $(3,1)/(2,1)$; the remaining boxes form a conjugate-semistandard tableau with maximum entry 3.

```
type L = Int

partitionChainToFamily :: PartitionChain → K → [L] → [Tableau]
partitionChainToFamily qs k ls = sortBy entryOrdering [putInPlusEntries lbss t | t ← ts]
    where ts = entryOrderCSSYTs (last qs) k
          bss = partitionChainToBoxChain qs
          lbss = zip ls bss

putInPlusEntries :: [(L, [Box])] → Tableau → Tableau
putInPlusEntries lbss t = tableauMToTableau $ putInPlusEntriesM lbss
                                            $ tableauToTableauM t

putInPlusEntriesM [] tM = tM
putInPlusEntriesM ((l, bs) : rest) tM = putInPlusEntriesM rest tM′
    where tM′ = insertMany tM [(b, l + 1) | b ← bs]
```

9.5. **Examples.**

(1) In Example 9.6 in [1], we find the lexicographically minimal conjugate-semistandard tableau family of shape $(3,1)^7$. At Step 4, we have $k_1 = 3$, $k_2 = 2$, $k_3 = 1$ and we require just one more tableau, and there are three partition chains:

$$\begin{aligned}&(3,1) \to_1 (3) \to_1 (2) \to_1 (1),\\ &(3,1) \to_1 (2,1) \to_1 (2) \to_1 (1),\\ &(3,1) \to_1 (2,1) \to_1 (1,1) \to_1 (1).\end{aligned}$$

Taking $k_4 = 1$ they give the three tableau below:

$$\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4\\\cline{1-1}\end{array}\,,\quad \begin{array}{|c|c|c|}\hline 1&2&4\\\hline 3\\\cline{1-1}\end{array}\,,\quad \begin{array}{|c|c|c|}\hline 1&3&4\\\hline 2\\\cline{1-1}\end{array}\,.$$

This is one too many, so $k_4 = 0$, and correspondingly *kPair* $[3,1]$ $[1,1,1]$ 2 evaluates to *Just* $(0,0)$. Exactly the same tableaux correspond to the chains $(3,1) \to_1 \cdots \to_1 \to \varnothing$, and since there is a unique empty conjugate-semistandard tableau, even taking $k_5 = 0$ gives too many tableau. Therefore *kPair* $[3,1]$ $[1,1,1,1]$ evaluates to *Nothing*. (This is the only 'failure' case.)

(2) We give a further example to show the case when $k_1 = k_2$; then the partition chains at Step 3 are given by the Pieri removal of two boxes from μ, reflecting that both will get the entry $k_1 + 1$. Take $\mu = (2,1)$ and $n = 7$. In Step 1, since there are 8 conjugate-semistandard tableau with maximum permitted entry 3, we take $k_1 = 2$, getting

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 1 \\ \cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 2 \\ \cline{1-1}\end{array}.$$

In Step 2 the chains $(2,1) \to_1 (2)$ and $(2,1) \to_1 (1,1)$ correspond to tableaux of the forms

$$\begin{array}{|c|c|}\hline \star & \star \\ \hline 3 \\ \cline{1-1}\end{array} \quad \begin{array}{|c|c|}\hline \star & 3 \\ \hline \star \\ \cline{1-1}\end{array}.$$

(Here $3 = k_1 + 1$ is inserted by the code immediately above.) Since $k_1 = 3$ was too big on Step 2, we have $k_2 \leq 2$, and taking $k_2 = 2$ gives

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline 3 \\ \cline{1-1}\end{array}, \quad \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 1 \\ \cline{1-1}\end{array}, \quad \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 \\ \cline{1-1}\end{array}, \quad \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 2 \\ \cline{1-1}\end{array}.$$

One more tableau is required, and in Step 3 we remove a Pieri chain of 2 boxes, and take $k_3 = 1$, getting

$$\begin{array}{|c|c|}\hline 1 & 3 \\ \hline 1 \\ \cline{1-1}\end{array}.$$

The algorithm, as coded below, continues with $k_4 = 0$ (and no tableaux are taken in the final step). After Step 2, the function *newCLS* below updates the tuple $(k_1) = (2) = (\ell_1^{c_1}) = (2^1)$ to $(k_1, k_2) = (2,2) = ({\ell'}_1^{c'_1}) = (2^2)$. The full output of the algorithm can be seen using *printAlg* \$ *lexMinimalFamilyAll* $[2,1]$ 7.

9.6. **Algorithm 9.5.**

```
oneStep :: Partition → [(NumberOfBoxesToRemove, L)] → Target → (K, [Tableau])
oneStep p cls target =
    case kPair p cs target of
        Just (k, a) → (k, combine [partitionChainToFamily qs k ls
                                        | (qs, _) ← chainsWithSizes p cs k])
        Nothing → (−1, combine [partitionChainToFamily qs (−1) ls
                                        | (qs, _) ← chainsWithSizes p cs (−1)])
    where cs = [c | (c, _) ← cls]
          ls = [l | (_, l) ← cls]
          combine = sortBy entryOrdering ∘ concat

newCLS :: [(NumberOfBoxesToRemove, L)] → K → [(NumberOfBoxesToRemove, L)]
newCLS [] k = [(1, k)]
```

$$
\begin{aligned}
&\mathit{newCLS}\ \mathit{cls}\ k \mid l \equiv k = \mathit{dropLast}\ 1\ \mathit{cls} +\!\!+ [(c+1,k)] \\
&\qquad\qquad\quad \mid \mathit{otherwise} = \mathit{cls} +\!\!+ [(1,k)] \\
&\quad \mathbf{where}\ (c,l) = \mathit{last}\ \mathit{cls}
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{oneStepFull} :: \mathit{Partition} \to [(\mathit{NumberOfBoxesToRemove}, L)] \to \mathit{Target} \\
&\qquad\qquad\quad \to ([(\mathit{NumberOfBoxesToRemove}, L)], \mathit{Target}, [\mathit{Tableau}]) \\
&\mathit{oneStepFull}\ p\ \mathit{cls}\ \mathit{target} = (\mathit{newCLS}\ \mathit{cls}\ k, \mathit{target} - \mathit{length}\ \mathit{ts}, \mathit{ts}) \\
&\quad \mathbf{where}\ (k, \mathit{ts}) = \mathit{oneStep}\ p\ \mathit{cls}\ \mathit{target}
\end{aligned}
$$

$$
\mathbf{type}\ \mathit{NumberOfSteps} = \mathit{Int}
$$

$$
\begin{aligned}
&\mathit{sSteps} :: \mathit{Partition} \to [(\mathit{NumberOfBoxesToRemove}, L)] \to \mathit{Target} \to \mathit{NumberOfSteps} \\
&\qquad\quad \to ([(\mathit{NumberOfBoxesToRemove}, L)], \mathit{Target}, [[\mathit{Tableau}]]) \\
&\mathit{sSteps}\ \_\ \mathit{cls}\ t\ 0 = (\mathit{cls}, t, [\,]) \\
&\mathit{sSteps}\ p\ \mathit{cls}\ t\ s = (\mathit{cls}'', t'', \mathit{ts} : \mathit{ts}') \\
&\quad \mathbf{where}\ (\mathit{cls}', t', \mathit{ts}) = \mathit{oneStepFull}\ p\ \mathit{cls}\ t \\
&\qquad\qquad (\mathit{cls}'', t'', \mathit{ts}') = \mathit{sSteps}\ p\ \mathit{cls}'\ t'\ (s-1)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{lexMinimalFamilyAll} :: \mathit{Partition} \to \mathit{Target} \\
&\quad \to ([(\mathit{NumberOfBoxesToRemove}, L)], [\mathit{NumberOfNewTableaux}], [[\mathit{Tableau}]], [\mathit{Tableau}]) \\
&\mathit{lexMinimalFamilyAll}\ p\ \mathit{target} = \\
&\quad \mathbf{let}\ (\mathit{cls}, \mathit{target}', \mathit{tss}) = \mathit{sSteps}\ p\ [\,]\ \mathit{target}\ (\mathit{sum}\ p) \\
&\qquad (\_, \mathit{ts}') = \mathit{oneStep}\ p\ \mathit{cls}\ \mathit{target}' \\
&\qquad \mathit{as} = \mathit{map}\ \mathit{length}\ \mathit{tss} +\!\!+ [\mathit{target}'] \\
&\qquad \mathbf{in}\ (\mathit{cls}, \mathit{as}, \mathit{tss}, \mathit{ts}')
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{allLexMinimalFamilies} :: \mathit{Partition} \to \mathit{Target} \to [[\mathit{Tableau}]] \\
&\mathit{allLexMinimalFamilies}\ p\ t = \mathbf{let}\ (\_, \mathit{as}, \mathit{tss}, \mathit{ts}') = \mathit{lexMinimalFamilyAll}\ p\ t \\
&\qquad\qquad\qquad\qquad\qquad\quad \mathbf{in}\ [\mathit{concat}\ \mathit{tss} +\!\!+ \mathit{ts} \mid \mathit{ts} \leftarrow \mathit{subsequencesOfLength}\ (\mathit{last}\ \mathit{as})\ \mathit{ts}']
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{finalChoices}\ p\ t = \mathit{subsequencesOfLength}\ (\mathit{last}\ \mathit{as})\ \mathit{ts}' \\
&\quad\ \mathbf{where}\ (\_, \mathit{as}, \mathit{tss}, \mathit{ts}') = \mathit{lexMinimalFamilyAll}\ p\ t
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{leastLexMinimalFamily} :: \mathit{Partition} \to \mathit{Target} \to [\mathit{Tableau}] \\
&\mathit{leastLexMinimalFamily}\ p\ t = \mathbf{let}\ (\_, \mathit{as}, \mathit{tss}, \mathit{ts}') = \mathit{lexMinimalFamilyAll}\ p\ t \\
&\qquad\qquad\qquad\qquad\qquad\quad \mathbf{in}\ \mathit{concat}\ \mathit{tss} +\!\!+ \mathit{take}\ (\mathit{last}\ \mathit{as})\ \mathit{ts}'
\end{aligned}
$$

For example, *printFamilies* \$ *allLexMinimalFamilies* $[2,1]$ 10 evaluates to

$$\ytableaushort{12,1},\ \ytableaushort{12,2},\ \ytableaushort{13,1},\ \ytableaushort{12,3},\ \ytableaushort{13,2},\ \ytableaushort{23,2},\ \ytableaushort{13,3},\ \ytableaushort{23,3},\ \ytableaushort{14,1},\ \ytableaushort{12,4}$$

$$\ytableaushort{12,1},\ \ytableaushort{12,2},\ \ytableaushort{13,1},\ \ytableaushort{12,3},\ \ytableaushort{13,2},\ \ytableaushort{23,2},\ \ytableaushort{13,3},\ \ytableaushort{23,3},\ \ytableaushort{14,1},\ \ytableaushort{14,2}$$

The first of these is the least family (the tie in the entry order is broken by the total column order). The two tableau in the final position that complete the families are the output of Step F of Algorithm 9.5, and can be constructed using *finalChoices* $[2,1]$ 10.

9.7. Printing output of Algorithm 9.5.

```haskell
printSteps :: ([(NumberOfBoxesToRemove, L)], [[Tableau]]) -> IO ()
printSteps (cls, tss) =
    do putStrLn $ show cls
       sequence_ [printTableaux ts | ts <- tss]

printAlg :: ([(NumberOfBoxesToRemove, L)], [NumberOfNewTableaux],
                        [[Tableau]], [Tableau]) -> IO ()
printAlg (cls, as, tss, ts') =
    do putStrLn $ show cls
       putStrLn $ show as ++ "\n"
       sequence_ [printTableaux ts >> putStrLn "" | ts <- tss, ts /= []]
       printTableaux ts'
```

Pretty printing of tableaux.

```haskell
printList :: (Show a) => [a] -> IO ()
printList xs = putStrLn $ unlines $ map show xs

printListMagma :: (Show a) => [a] -> IO ()
printListMagma xs = putStrLn $ "[" ++ (dropLast 2 $ unlines $ [show x ++ "," | x <- xs]) ++ "]\n"


showTableau :: Tableau -> String
showTableau t = concat [showTableauRow es ++ "\n" | es <- t]

showTableauRow :: [Entry] -> String
showTableauRow es = concat [f e | e <- es] ++ " "
        where f 0 = "."; f 10 = "T"; f 11 = "J"; f 12 = "Q"; f 13 = "K";
              f 14 = "A"; f 15 = "F"; f e = show e

printTableauxNoLn :: [Tableau] -> IO ()
printTableauxNoLn ts = putStr $ showTableaux ts
```

$printTableaux :: [\,Tableau\,] \to IO\ ()$
$printTableaux\ ts = putStrLn\ \$\ showTableaux\ ts$

$showTableaux :: [\,Tableau\,] \to String$
$showTableaux\ [\,] = \texttt{""}$
$showTableaux\ ts \mid length\ ts \leqslant 10 = ls$
$\qquad\qquad\qquad \mid otherwise = ls +\!\!+ \texttt{"\textbackslash n"} +\!\!+ showTableaux\ (drop\ 10\ ts)$
$\qquad \textbf{where}\ ls = unlines\ (linesTableauxB\ (take\ 10\ ts))$

$linesTableauPadding\ t = [\,pad\ (s - length\ l)\ l \mid l \leftarrow ls\,]$
$\qquad \textbf{where}\ ls@(l : \_) = lines\ (showTableau\ t)$
$\qquad\qquad\quad s = length\ l + 2$

$linesTableauxB\ ts = [\,concat\ l \mid l \leftarrow ls'\,]$
$\qquad \textbf{where}\ ls' = transpose\ [\,linesTableauPadding\ t \mid t \leftarrow ts\,]$

$pad\ s\ l = l +\!\!+ take\ s\ spaces$
$\qquad \textbf{where}\ spaces = repeat\ \texttt{' '}$

$printFamilies\ tss = sequence\_\ [\,printTableaux\ ts \gg putStrLn\ \texttt{""} \mid ts \leftarrow tss\,]$

10. Utility functions

$xs\ `diff`\ ys = [\,x \mid x \leftarrow xs, \neg\ (x \in ys)\,]$

$xs\ `meet`\ ys = [\,x \mid x \leftarrow xs, x \in ys\,]$

$partialSums :: (Num\ a) \Rightarrow [\,a\,] \to [\,a\,]$
$partialSums = scanl1\ (+)$

$fromJust\ (Just\ x) = x$
$fromJust\ Nothing = error\ \texttt{"fromJust: Nothing"}$

$maybeLast\ [\,] = Nothing$
$maybeLast\ xs = Just\ (last\ xs)$

$position\ x\ xs = fromJust\ \$\ lookup\ x\ (zip\ xs\ [0\mathinner{\ldotp\ldotp}])$

$dropLast\ k\ xs = reverse\ \$\ drop\ k\ \$\ reverse\ xs$

$subsequencesOfLength\ 0\ \_ = [[\,]]$
$subsequencesOfLength\ \_\ [\,] = [\,]$
$subsequencesOfLength\ k\ (y : ys) =$

$$[\,y : ys \mid ys \leftarrow \mathit{subsequencesOfLength}\ (k-1)\ ys\,] +\!\!+\ \mathit{subsequencesOfLength}\ k\ ys$$

$$\begin{aligned}
&\mathit{collectSorted}\ [\,] = [\,]\\
&\mathit{collectSorted}\ (x : [\,]) = [\,(x,1)\,]\\
&\mathit{collectSorted}\ (x : ys) = (x,m) : \mathit{collectSorted}\ ys'\\
&\qquad\qquad \textbf{where}\ m = 1 + \mathit{length}\ (\mathit{takeWhile}\ (\equiv x)\ ys)\\
&\qquad\qquad\qquad\quad ys' = \mathit{dropWhile}\ (\equiv x)\ ys
\end{aligned}$$

References

[1] Rowena Paget and Mark Wildon, *Generalized Foulkes modules and maximal and minimal constituents of plethysms of Schur functions*, arXiv:1608.04018 (2016), 42 pages.