Ruhr-Universität Bochum

Fakultät für Mathematik
Algebraische Kombinatorik

Tableau combinatorics and insertion algorithms

A thesis submitted for the degree of
Master of Science

author
Tim Strunkheide

student number
108016215942

supervisor
Prof. Dr. Christian Stump

secondary supervisor
PD Dr. Björn Schuster

March 17, 2023

Contents

Acknowledgements
I would like to express my gratitude to Christian Stump for his excellent guidance and very kind support throughout this project. I am also want to thank Eric Marberg for fruitful discussions, Galen Dorpalen-Barry for her expert assistance in almost every matter, and Elena Hoster for always being there to listen and offer support. Thank you all for helping me complete this thesis.

1 Introduction

The Schur P-functions and Schur Q-functions are well known families of symmetric functions. Each of them are a basis of the space of symmetric functions, have applications in respresentation theory and algebraic combinatorics and they generate a ring with non-negative structure constants. The latter fact was also conjectured for their K-theoretic analogues GP_λ and GQ_λ by Ikeda and Naruse in 2013 in [1]. In 2014, Clifford, Thomas and Yong proved this for GP_λ in [2]. However, it is still unproven for GQ_λ.

In this thesis we look onto the symplectic Hecke insertion and a semistandard variant of it, which was introduced by Eric Marberg in [5]. In this thesis we present some of the properties of the symplectic Hecke insertion and use them to proof its bijectivity. With this bijection we are able to understand the bijective approach to proof that K-theoretic Schur P-functions GP_λ generate a ring, which was outlined by Eric Marberg in [4]. The main contribution of this thesis is the correction of an error in the definition of weakly admissible tableaux [5, Definition 3.5].

In a former version of it a tableau with an outer row box was weakly admissible, if either $i = 1$ or there exists a column $x \geq i$ with $T_{i-1,x} \leq T_{in} < T_{ix}$. So the last part of the definition was missing. As a result, the first statement of Lemma 3.6 was wrong and as a result also Theorem 3.8.

A closer look onto the details of the proof of Lemma 3.6 reveals the problem: in the case where we have the outer box not in the first row, we are not able to imply that $c_2 \neq b$ if $k = 0$. We are not able to evade this, since there are counterexamples, where the lemma is wrong with the former definition, for example:

$$\begin{array}{|c|c|}\hline 1 & 2 \\ \hline \end{array}\ \cdot\ \cdot\ \cdot\ \begin{array}{|c|}\hline 2 \\ \hline \end{array} \underset{(R2)}{\longrightarrow} \begin{array}{|c|c|}\hline 1 & 2 \\ \hline \end{array}$$

with row reading word 212 and 12 respectively, but $212 \overset{\mathrm{K}}{\not\approx} 12$, which contradicts the lemma.

Eric Marberg and I looked for a solution to this and corrected the definition by including the missing condition from the definition of admissibility to the definition for weakly admissibility. With the new version of weakly admissibility, the former proof holds. In our example, the first insertion state is not weakly admissible anymore, so the lemma holds. Also, the other parts of [5], where weakly admissibility is used, still hold.

In Chapter 2, we introduce the objects we need to understand the symplectic Hecke insertion. The main contribution of this thesis is Chapter 3, where we introduce the algorithm for symplectic Hecke insertion. Therein, Definition 3.3 describes the main algorithm while Theorem 3.1 is the main result of this thesis. In section 3.3 we introduce a semistandard variant of the insertion algorithm and prove its most important properties in Theorem 3.29. In Chapter 4, we use the previous results to proof that the K-theoretic Schur P-functions have positive structure constants in Theorem 4.1. The last chapter 5 of this thesis presents Sage code that implements the insertion algorithms, their inverses, and other methods introduced in the previous chapters.

2 Preliminaries

In this paper we refer to the positive integers as $\mathbb{N}$ and to the non-negative integers as $\mathbb{N}^0$. Let $\mathcal{S}_n$ be the group of permutations on $[n] := \{i \in \mathbb{N} : i \leq n\}$. We define $\mathcal{S}_\infty$ as the group of permutations with finite support, that means $\mathcal{S}_\infty = \bigcup_{n\in\mathbb{N}} \mathcal{S}_n$. The group $\mathcal{S}_n$ is generated by $\{(i, i+1) : 1 \leq i \leq n-1\}$, the set of permutations, which swap adjacency positions. We denote $s_i = (i, i+1)$, so $\mathcal{S}_n = \langle s_1, s_2, \dots, s_{n-1}\rangle$ and $\mathcal{S}_\infty := \langle s_1, s_2, \dots\rangle$.

We use this chapter to define the fundamental objects we require later. Especially, we introduce symplectic Hecke words, equivalences on words and tableaux.

2.1 Words

A **word** w is a finite sequence of numbers $w = w_1 w_2 \dots w_k$ with $w_i \in \mathbb{N}$. The numbers w_i are called **letters**. The **length** of w is $len(w) = k$.

We define now the set of Hecke words. Although, we do not use them in this paper, we need their related symplectic anaglogue. Define a map $\circ : \mathcal{S}_\infty \times \{s_1, s_2, \dots\} \longrightarrow \mathcal{S}_\infty$ by

$$\pi \circ s_i = \begin{cases} \pi s_i, \text{ if } \pi(i) < \pi(i+1), \\ \pi, \text{ if } \pi(i) > \pi(i+1). \end{cases}$$

We define a **Hecke word** for a permutation σ as a word $w = w_1 w_2 \dots w_k$, such that $\sigma = w_1 \circ w_2 \circ \dots w_k$. A Hecke word for σ with minimal length is called **reduced word** for σ. The set of reduced words for σ is called $\mathcal{R}(\sigma)$.

In this paper, we work with fix-point-free involutions. We define the set of **fix-point-free involutions** as $\mathcal{F}_\infty := \{\pi^{-1}\Theta\pi : \pi \in \mathcal{S}_\infty\}$, where Θ is the permutation $\Theta : \mathbb{N} \longrightarrow \mathbb{N}$, $\Theta(n) = n - (-1)^n$. Note, that neither Θ nor any element of $\mathcal{F}_\infty$ is in $\mathcal{S}_\infty$, because they do not have finite support, since $\Theta = \prod_{i \text{ odd}} s_i = (12)(34)(56)(78)\dots$. We define $\mathcal{F}_n := \{z \in \mathcal{F}_\infty : z(i) = \Theta(i) \text{ for } i > n\}$, so $\mathcal{F}_\infty = \bigcup_{n\in\mathbb{N}} \mathcal{F}_n$.

We define a map $\odot : \mathcal{F}_\infty \times \{s_1, s_2, \dots\} \longrightarrow \mathcal{F}_\infty \cup \{0\}$ by

$$z \odot s_i = \begin{cases} s_i z s_i & , \text{ if } z(i) < z(i+1), \\ z & , \text{ if } i+1 \neq z(i) > z(i+1) \neq i, \\ 0 & , \text{ if } i+1 = z(i) > z(i+1) = i. \end{cases}$$

A **symplectic Hecke word** for $z \in \mathcal{F}_\infty$ is a word $w_1 w_2 \dots w_k$, such that it holds that $z = \Theta \odot s_{w_1} \odot s_{w_2} \odot \dots \odot s_{w_k}$. We denote the set of all symplectic Hecke words for a fix-point-free involution $z \in \mathcal{F}_\infty$ as $\mathcal{H}_{\mathrm{Sp}}(z)$. A symplectic Hecke word in $\mathcal{H}_{\mathrm{Sp}}(z)$ of minimal length is called **FPF-involution word** for z and the set of FPF-involution words for z is called $\mathcal{R}_{FPF}(z)$.

Remark 2.1. Since $\Theta \odot s_i = 0$ if i is odd, every symplectic Hecke word begins with an even letter. If $w = w_1 w_2 \dots w_k$ is a symplectic Hecke word, then $w = w_1 w_2 \dots w_i w_i \dots w_k$ is also one.

We can conclude from the definition, that every subword $w_i w_{i+1} \dots w_{i+k}$ of an FPF-involution word w has to be an FPF-involution word, or else the former word w would not be an FPF-involution word.

Example 1. Let $w = 26542$. It holds, that

$$\begin{aligned}\theta \odot s_2 \odot s_6 \odot s_5 \odot s_4 \odot s_2 &= (23)\theta(23) \odot s_6 \odot s_5 \odot s_4 \odot s_2 \\ &= (67)(23)\theta(23)(67) \odot s_5 \odot s_4 \odot s_2 \\ &= (56)(67)(23)\theta(23)(67)(56) \odot s_4 \odot s_2 \\ &= (45)(56)(67)(23)\theta(23)(67)(56)(45) \odot s_2 \\ &= (45)(56)(67)(23)\theta(23)(67)(56)(45) \\ &= (13)(25)(48)(67)(9,10)(11,12)\cdots =: \check{z},\end{aligned}$$

So $26542 \in \mathcal{H}_{\mathrm{Sp}}(\check{z})$ und $\check{z} \in \mathcal{F}_8$. Furthermore, $2654 \in \mathcal{R}_{\mathrm{FPF}}(\check{z}) \subset \mathcal{H}_{\mathrm{Sp}}(\check{z})$.

We know, that 526 is not a symplectic Hecke word for any fix-point-free involution, since it starts with an odd letter. With 26542, there is also $265542 \in \mathcal{H}_{\mathrm{Sp}}(\check{z})$. Since 265 is a subword of 2654, it follows that 265 is an FPF-involution word. And indeed, $256 \in \mathcal{R}_{\mathrm{FPF}}((56)(67)(23)\theta(23)(67)(56))$.

Let $w = w_1 w_2 \dots w_m$ be a word of length m. A **weakly increasing factorisation** of w is a weakly increasing sequence of positive integers $i = (i_1 \leq i_2 \leq \cdots \leq i_m) \in \mathbb{N}^m$ with $i_j \leq i_{j+1}$ if $w_j > w_{j+1}$. The **weight** of such a factorisation is the map $\mathrm{wt}_{(w,i)} : \mathbb{N} \to \mathbb{N}^0$ with $wt_{(w,i)}(a) = |\{j \in [m] : i_j = a\}|$ for $a \in \mathbb{N}$.

The set of all pairs (w, i), where i is a weakly increasing factorisation of w, is called $\mathrm{Incr}(w)$. If W is a set of words, we denote $\mathrm{Incr}(W) = \bigcup_{w \in W} \mathrm{Incr}(w)$.

Example 2. Every symplectic Hecke word of length n has $(1, 2, 3, 4, \dots, n)$ as weakly increasing factorisation.

We already saw some symplectic Hecke words in the previous example, so we can now define weakly increasing factorisations for them:

$$(26542, 11234), (26542, 12345), (26542, 22578), (2654, 1137) \in \mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(\check{z})).$$

We take the first of them and calculate

$$wt_{(26542,11234)}(2) = wt_{(26542,11234)}(3) = wt_{(26542,11234)}(4) = 1$$

and

$$wt_{(26542,11234)}(1) = 2.$$

A weakley increasing factorisation splits a word into increasing parts. We can summarize the notation of a word with a weakly increasing factorisation as follows: we write $(w^1, w^2, w^3, \dots)$, where $w = w^1 w^2 w^3 \dots$ and $w^i = w_k w_{k+1} \dots w_{k+l}$ is the subword with all the indices, so that $i_k = i_{k+1} = \cdots = i_{k+l}$. By the definition of weakly increasing factorisations, the words w^i has to be increasing. We also use the term "weakly increasing factorisation" for these tuples $(w^1, w^2, w^3, \dots)$ of possible empty increasing subwords of w, with $w = w^1 w^2 w^3 \dots$, whereby all but finitely many w^i has to be empty. Moreover, we abuse the introduced notation and write $(w, i) = (w^1, w^2, w^3, \dots) \in \mathrm{Incr}(w)$. By the definition, we can see that $wt_{(w,i)}(j) = \mathrm{len}(w^j)$.

Example 3. With the notation of the previous examples, we can write now

$$(26542, 11234) = (26, 5, 4, 2) \in \mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(\check{z})).$$

Let $y \in \mathcal{F}_m, z \in F_n$ be two fix-point-free involutions. We define another fix-point-free involution $y \times z \in \mathcal{F}_{m+n}$ by $y \times z : i \mapsto y(i), m + j \mapsto z(j) + m$ for $1 \leq i \leq m, 1 \leq j \leq n$, and also a bijection

$$\begin{aligned} \mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(y)) \times \mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(z)) &\longrightarrow \mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(y \times z)) \\ (v_1, v_2, \dots) \times (w_1, w_2, \dots) &:= (v_1 \check{w_1}, v_2 \check{w_2}, \dots), \end{aligned}$$

where $\check{w_i}$ is formed by adding m to every letter of w_i.

We can see directly, that $\mathrm{wt}_{(v_1,v_2,\dots)} + \mathrm{wt}_{(w_1,w_2,\dots)} = \mathrm{wt}_{(v_1\check{w_1},v_2\check{w_2},\dots)}$.

2.2 Equivalences

We define some useful and known relations on words:

Define $\overset{\mathrm{Br}}{=}$, the **braid relation**, as the equivalence relation on words generated by the following rules: for every word a, b and all $X, Y \in \mathbb{N}$ with $|X - Y| > 1$ it holds that

- $a(X+1)X(X+1)b \overset{\mathrm{Br}}{=} aX(X+1)Xb$ and
- $aXYb \overset{\mathrm{Br}}{=} aYXb$.

The equivalence relation $\overset{\mathrm{Br}}{\equiv}$ is defined similarly as the equivalence relation generated by the rules: for all words a, b, and all $X, Y, Z \in N$ with $|X - Y| > 1$ it holds that

- $aXZXb \overset{\mathrm{Br}}{\equiv} aZXZb$,
- $aXYb \overset{\mathrm{Br}}{\equiv} aYXb$ and
- $aXb \overset{\mathrm{Br}}{\equiv} aXXb$

Define the symplectic variant $\overset{\mathrm{Sp}}{=}$ to be the equivalence relation on words generated by $v \overset{\mathrm{Sp}}{=} w$, if $v \overset{\mathrm{Br}}{=} w$, and $X(X-1)a \overset{\mathrm{Sp}}{=} X(X+1)a$ for all words a, v, w and all $X \geq 2$. Similarly, define $\overset{\mathrm{Sp}}{\equiv}$ to be the equivalence relation on words, with $v \overset{\mathrm{Sp}}{\equiv} w$, if $v \overset{\mathrm{Br}}{\equiv}$, and $X(X-1)a \overset{\mathrm{Sp}}{\equiv} X(X+1)a$ for all words a, v, w and $X \in \mathbb{N}$ with $X \geq 2$

Lemma 2.2. If $z \in \mathcal{F}_\infty$ is a fix-point-free involution, then $\mathcal{R}_{\mathrm{FPF}}(z)$ is an equivalence class under $\overset{\mathrm{Sp}}{=}$, while $\mathcal{H}_{\mathrm{Sp}}(z)$ is an equivalence class under $\overset{\mathrm{Sp}}{\equiv}$. A word is a symplectic Hecke word, if and only if its equivalence class under $\overset{\mathrm{Sp}}{\equiv}$ does not contain a word that begins with an odd leter. A symplectic Hecke word is an FPF-involution word, if and only if its equivalence class under $\overset{\mathrm{Sp}}{=}$ contains no word with equal adjacent letters.

Example 4. Starting with the word $26542 \in \mathcal{H}_{\mathrm{Sp}}(\check{z})$, which we found in Example 1. We can find other symplectic Hecke words for $\check{z}$ by the following computing:

$$26542 \overset{\mathrm{Br}}{=} 62542 \overset{\mathrm{Br}}{=} 65242 \overset{\mathrm{Br}}{=} 65422 \overset{\mathrm{Br}}{=} 6542 \overset{\mathrm{Sp}}{=} 6742 \overset{\mathrm{Br}}{=} 6472 \overset{\mathrm{Br}}{=} 6427,$$

so $\{2654, 6254, 6524, 6542, 6742, 6472, 6427\} \subset \mathcal{H}_{\mathrm{Sp}}(\check{z})$.

Starting with $265 \in \mathcal{R}_{\mathrm{FPF}}((56)(67)(23)\theta(23)(67)(56))$, which we found in the same example, we get

$$265 \overset{\mathrm{Br}}{=} 625 \overset{\mathrm{Br}}{=} 652 \overset{\mathrm{Sp}}{=} 672 \overset{\mathrm{Br}}{=} 627 \overset{\mathrm{Br}}{=} 267$$

and since there is no other word which is $\overset{\mathrm{Sp}}{=}$-equivalent to any of this words, we get $\{265, 625, 652, 672, 627, 267\} = \mathcal{R}_{\mathrm{FPF}}((56)(67)(23)\theta(23)(67)(56))$.

Another family of relations is based on the **Coxeter-Knuth equivalence** $\overset{\mathrm{K}}{\sim}$ (resp. on the **K-Knuth equivalence** $\overset{\mathrm{K}}{\approx}$). This are the equivalence relations, generated by

$$aZXYb \overset{\mathrm{K}}{\sim} aXZYb, \quad aYZXb \overset{\mathrm{K}}{\sim} aYXZb, \quad aXYXb \overset{\mathrm{K}}{\sim} aYXYb,$$

resp.

$$aZXYb \overset{\mathrm{K}}{\approx} aXZYb, \quad aYZXb \overset{\mathrm{K}}{\approx} aYXZb, \quad aXYXb \overset{\mathrm{K}}{\approx} aYXYb, \quad aXb \overset{\mathrm{K}}{\approx} aXXb.$$

for all words a, b and every $X, Y, Z \in \mathbb{N}$ with $X < Y < Z$.

Remark 2.3. The first two rules for each of the latter equivalences state the following: in a sequence of three distinct letters, we can swap the lowest and the greatest, if they are adjacent.

To define a symplectic version of these equivalences we want to add another kind of rule: We say that two words are connected by a **symplectic Coxeter-Knuth move**, if one word is obtained from the other in one of these ways:

- By interchanging the first two letters, when they have the same parity.
- By changing the first two letters from $X(X-1)$ to $X(X+1)$ for an $X \in \mathbb{N}$ with $X \geq 2$.

We define the **symplectic Coxeter-Knuth equivalence** $\overset{\mathrm{Sp}}{\sim}$ as the strongest equivalence relation that has $v \overset{\mathrm{Sp}}{\sim} w$, if $v \overset{\mathrm{K}}{\sim} W$, or if v and w are connected by a symplectic Coxeter-Knuth move. Analogue, we define the **symplectic K-Knuth equivalence** $\overset{\mathrm{Sp}}{\approx}$ as the strongest equivalence relation with $v \overset{\mathrm{Sp}}{\approx} w$, if $v \overset{\mathrm{K}}{\approx} w$, or if v and w are connected by a symplectic Coxeter-Knuth move.

By looking onto the definitions of the equivalece relations, we can see that the following lemma holds.

Lemma 2.4. Let v, w be two words, $z \in \mathcal{F}_\infty$ a fix-point-free involution. If $v \in \mathcal{H}_{\mathrm{Sp}}(z)$ and $v \overset{\mathrm{Sp}}{\approx} w$, then $w \in \mathcal{H}_{\mathrm{Sp}}(z)$. If $v \in \mathcal{R}_{\mathrm{FPF}}(z)$ and $v \overset{\mathrm{Sp}}{\sim} w$, then $w \in \mathcal{R}_{\mathrm{FPF}}(z)$.

Example 5. We can see, that

$$62467 \overset{\mathrm{Sp}}{\sim} 26467 \overset{\mathrm{K}}{\sim} 24647 \overset{\mathrm{K}}{\sim} 24674 \overset{\mathrm{Sp}}{\sim} 42674 \overset{\mathrm{K}}{\sim} 46274 \overset{\mathrm{K}}{\sim} 46724$$
$$\overset{\mathrm{Sp}}{\sim} 64724 \overset{\mathrm{K}}{\sim} 67424 \overset{\mathrm{Sp}}{\sim} 65424 \overset{\mathrm{K}}{\sim} 65242 \overset{\mathrm{K}}{\sim} 62542 \overset{\mathrm{Sp}}{\sim} 26542.$$

And since the last word is the word, we took in Example 1, all the listed words are symplectic Hecke words in $\mathcal{H}_{\mathrm{Sp}}(\check{z})$.

2.3 Tableaus

In this part we introduce a few definitions about partitions, Young diagrams and tableaux.

Let n be a non-negative integer. A tupel $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_k)$ of positive integers $\lambda_i \in \mathbb{N}$ with $\sum \lambda_i = n$ and $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k$ is a **partition** of n. The length of a partition $\mathrm{len}((\lambda_1, \lambda_2, \ldots, \lambda_k))$ is k. A partition λ is **strict**, if $\lambda_1 > \lambda_2 > \cdots > \lambda_k$.

Example 6. A partiton of 5 is $(2, 2, 1)$, while a strict partition of 5 is $(3, 2)$.

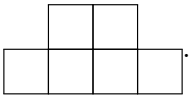
All possible strict partitions of 6 are $(6), (5, 1)$ and $(4, 2)$.

A **Young diagram** is a subset of $\mathbb{N} \times \mathbb{N}$. The Young diagram of a partition λ is the subset $\mathcal{D}_\lambda = \{(i, j) : 1 \leq j \leq \lambda_i\}$. The shifted Young diagram of a strict partition λ is the subset $\mathcal{SD}_\lambda = \{(i, j) : i \leq j \leq i - 1 + \lambda_i\}$.
To visualise a Young diagram we use a lattice of boxes: for every $(i, j) \in Y$ we draw a box in the i-th row and the j-th column. For this paper we use the French notation, so we start counting the rows from bottom to top and the rows from left to right.

Example 7. The shifted Young diagram for the partition $(4, 2)$ is

$$\mathcal{SD}_{(4,2)} = \{(1, 1), (1, 2), (1, 3), (1, 4), (2, 2), (2, 3)\},$$

so the visualization is



A **tableau** T is a map from a Young diagram Y to $\mathbb{N}$. We write $(i, j) \in T$, if $(i, j) \in Y$ and set $T_{ij} = T(i, j)$ for every $(i, j) \in Y$. The **shape** of a tableau is the underlying Young diagram. A **shifted** tableau is a tableau, which shape is a shifted Young diagram $\mathcal{SD}_\lambda$ for any strict partition λ. To visualise a tableau we take the visualisation of the underlying Young diagram and write T_{ij} in the box at position (i, j).

Example 8. We take a shifted tableau and its visualisation:

$$\begin{aligned} T : \mathcal{SD}_{(4,2)} &\longrightarrow \mathbb{N}, \\ (1, 1) &\longmapsto 2, \\ (1, 2) &\longmapsto 4, \\ (1, 3) &\longmapsto 5, \\ (1, 4) &\longmapsto 6, \\ (2, 2) &\longmapsto 6, \\ (2, 3) &\longmapsto 7. \end{aligned}$$

T=

	6	7	
2	4	5	6

We identify tableaux and Young diagramms with their visualisations.

The **main diagonal** of a tableau T are the boxes at the positions (i, i) (or the restriction of T to $\{(i, i) : i \in \mathbb{N}\} \cap Y$).

For a tableau T we define the **row reading word** by concatenating the entries in the tableau, starting at first entry in the topmost row and continuing ("like reading") from left to right and from top to bottom. For the **column reading word** we start at the highest entry in the first column and continuing from top to bottom and from left to right.

Example 9. We take the tableau from the previous example and see, that

$$\mathrm{row}(T) = 672456, \quad \mathrm{col}(T) = 264756$$

and the main diagonal are the first boxes of each row.

A tableau is called **increasing**, if for two distinct boxes $(x, y) \neq (a, b)$ it holds that $T_{xy} < T_{ab}$ if and only if $x \leq a$ and $y \leq b$.

Example 10. We take a look onto the following two tableaux:

	6	7	
2	4	5	6

	6	7
2	2	5	6

The first tableau is increasing, the second one is not, since $T_{11} = T_{12}$.

The **length** $|T|$ of a tableau T is the number of boxes it occupies. We call a tableau a **standard** tableau, if it is increasing and its range is $[|T|]$.

Example 11. We list all the shifted standard tableaux of shape $(4, 2)$:

	5	6	
1	2	3	4

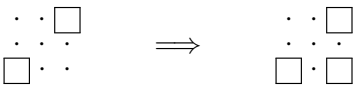
	4	6	
1	2	3	5

	4	5	
1	2	3	6

	3	5	
1	2	4	6

	3	6	
1	2	4	5

Since tableaux were defined onto arbitrary subsets of $\mathbb{N} \times \mathbb{N}$, we take a short look onto tableaux, which are not defined on Young diagrams of a partition. Especially, their visualisation could have holes. A tableau is called **row-column-closed**, if for every $(a, b), (x, y) \in T$ with $a \leq x$ and $b \leq y$, it holds that $(a, y) \in T$. The following picture illustrates this condition.



Lemma 2.5. If T is an increasing row-column-closed tableau, then $\mathrm{row}(T) \overset{\mathrm{K}}{\sim} \mathrm{col}(T)$.

Proof. By induction over the number of columns of T:

If T has only one column, $\mathrm{row}(T) = \mathrm{col}(T)$.

So let T be an increasing row-column-closed tableau with j columns.

Form the word w by reading the last column of T in reverse order (so from highest to lowest entry). Form U from T by removing the last column. Then U is also increasing and shifted and $\mathrm{col}(T) = \mathrm{col}(U)w$. By induction, $\mathrm{row}(U) \overset{\mathrm{K}}{\sim} \mathrm{col}(U)$, so we have to show now, that $\mathrm{row}(T) \overset{\mathrm{K}}{\sim} \mathrm{row}(U)w$. Let us say, that w is the j-th column and the highest entry in this column is the i-th, so $w = w_i w_{i-1} \dots w_2 w_1$ with $w_r = T_{rj}$.

So T has the form

$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	
T_{i1}	T_{i2}	$\dots$	T_{ii}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	T_{ij}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
T_{21}	T_{22}	T_{23}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	T_{2j}
T_{11}	T_{12}	T_{13}	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	$\cdots$	T_{1j}

but with possible several entries missing. Since T has to be row-column-closed, we know that for any non -empty row k, with $k < i$, (k, j) has to be occupied by T.

We get

$$\mathrm{row}(T) = \dots T_{i1} T_{i2} \dots T_{i,j-1} w_i T_{i-1,1} \dots T_{i-1,j-1} w_{i-1} \dots T_{11} T_{12} \dots T_{1,j-1} w_1,$$

while

$$\mathrm{row}(U)w = \dots T_{i1} T_{i2} \dots T_{i,j-1} T_{i-1,1} \dots T_{i-1,j-1} \dots T_{11} T_{12} \dots T_{1,j-1} w_i \dots w_2 w_1.$$

We only have to sort the letters of w from $\mathrm{row}(U)w$ to their place in $\mathrm{row}(T)$. So we have to show, that for every $k < i$ we can swap the k-th row for of the tableau with every w_l, where $l > k$. Since T is increasing, we know that, if the entries are existing, $T_{kk} < T_{k,k+1} < \dots < T_{k,j-1} < w_s$ for all $k \leq s$ and $w_1 < w_2 < \dots < w_{i-1} < w_i$.

So $T_{ks} w_t w_{t-1} \overset{\mathrm{K}}{\sim} w_t T_{ks} w_{t-1}$ for every $t > k$. By using this, we can pull $T_{1,j-1}$ through the letters of w to his place in $\mathrm{row}(T)$:

$$\begin{aligned}
\mathrm{row}(U)w &= \dots T_{22} T_{23} \dots T_{2,j-1} T_{11} T_{12} \dots T_{1,j-3} T_{1,j-2} T_{1,j-1} w_i w_{i-1} \dots w_2 w_1 \\
&\overset{\mathrm{K}}{\sim} \dots T_{22} T_{23} \dots T_{2,j-1} T_{11} T_{12} \dots T_{1,j-3} T_{1,j-2} w_i T_{1,j-1} w_{i-1} \dots w_2 w_1 \\
&\overset{\mathrm{K}}{\sim} \dots T_{22} T_{23} \dots T_{2,j-1} T_{11} T_{12} \dots T_{1,j-3} T_{1,j-2} w_i w_{i-1} \dots w_2 T_{1,j-1} w_1
\end{aligned}$$

Since every entry in the first row is lower than $T_{1,j-1}$, the same holds for every entry in the first row, so

$$row(U) \overset{\mathrm{K}}{\sim} \dots T_{22} T_{23} \dots T_{2,j-1} w_i w_{i-1} \dots w_2 T_{11} T_{12} \dots T_{1,j-3} T_{1,j-2} T_{1,j-1} w_1.$$

Now we can apply this to the other rows, by swapping the r-th row and every w_t with $r < t$.

$$\begin{aligned}
row(U)w &\overset{\mathrm{K}}{\sim} \dots w_i w_{i-1} \dots w_3 T_{22} T_{23} \dots T_{2,j-1} w_2 T_{11} T_{12} \dots T_{1,j-3} T_{1,j-2} T_{1,j-1} w_1 \\
&\overset{\mathrm{K}}{\sim} \dots T_{ii} T_{i,i+1} \dots T_{i,j-1} T_{i-1,i-1} \dots T_{i-1,j-1} w_i w_{i-1} \dots w_2 T_{11} T_{12} \dots w_1.
\end{aligned}$$

We can repeat this, till we reach the $(i-1)$-th row. First we use again $T_{i-1,j-1}w_iw_{i-1} \overset{\mathrm{K}}{\sim} w_iT_{i-1,j-1}w_{i-1}$. Now we need a slightly different argument, since we only have one element of w left. We can swap the other entries of the first row $T_{i-1,r}$ and w_i anyway, since $T_{i-1,r} < T_{i-1,r+1} < w_i$, and so $T_{i-1,r}w_iT_{i-1,r+1} \overset{\mathrm{K}}{\sim} w_iT_{i-1,r}T_{i-1,r+1}$ for every $r < j-1$. By using this repeatedly, we can swap now w_i with every entry of the $(i-1)$-th row and get

$$\begin{aligned} row(U)w &\overset{\mathrm{K}}{\sim} \dots T_{ii}T_{i,i+1}\dots T_{i,j-1}T_{i-1,i-1}\dots T_{i-1,j-2}w_iT_{i-1,j-1}w_{i-1}\dots \\ &\overset{\mathrm{K}}{\sim} \dots T_{ii}T_{i,i+1}\dots T_{i,j-1}T_{i-1,i-1}\dots T_{i-1,j-3}w_iT_{i-1,j-2}T_{i-1,j-1}w_{i-1}\dots \\ &\overset{\mathrm{K}}{\sim} \dots T_{ii}T_{i,i+1}\dots T_{i,j-1}w_iT_{i-1,i-1}\dots T_{i-1,j-2}T_{i-1,j-1}w_{i-1}\cdots = row(T). \end{aligned}$$

This completes the proof. □

Example 12. If we take a tableau, which is not row-column-closed, the statement can be wrong. If we take T as

·	·	3
1	2	·

then $\mathrm{row}(T) = 312 \overset{\mathrm{K}}{\sim} 132$. But $\mathrm{col}(T) = 123$.

Since shifted tableaux are row-column-closed, we can conclude the following lemma.

Theorem 2.6. If T is an increasing shifted tableau, then

$$\mathrm{col}(T) \overset{\mathrm{K}}{\sim} \mathrm{row}(T).$$

We define for every number $n \in \mathbb{N}$ a primed number $n' := n - \frac{1}{2}$. So we get the **primed alphabet** $\mathbb{P} = \{1' < 1 < 2' < 2 < \dots\}$. A map from a Young diagram Y to $\mathbb{P}$ is called **marked**. A map from a Young diagram Y to the power set of the primed numbers $\mathbb{P}$ is called a **set-valued** tableau. A map from a Young diagram Y to the set of finite, non-empty multisubsets of $\mathbb{P}$ is called a **weak set-valued** tableau.

Example 13. We look onto the following tableaux:

$T =$

	3	
1	$2'$	$4'$

$S =$

	36	
1	3	$4'$

$U =$

	33	
$1'$	1	$4'4'$

The first one is marked, the second one is set-valued and the third one is weak set-valued.

Many definitions for tableaux remain intuitively the same for (weak) set-valued tableaux, especially the ones which refer to the underlying Young diagram; so the shape of a set-valued tableau is also the underlying Young diagram and it is also shifted, if the shape is any $\mathcal{SD}_\lambda$.

The **length** of a (weak) set-valued tableau is the sum of the cardinality of all the sets in the tableau and it is denoted by $|T| := \sum_{i,j} |T_{ij}|$. The **weight** of a weak set-valued tableau T is the map $\mathrm{wt}(T) : \mathbb{N} \to \mathbb{N}^0$, whose value at $i \in \mathbb{N}$ is the number of times i or i' appears in T.

Example 14. By taking the tableaux of the previous example, we get $|S| = 5,\ |U| = 6$ and the weight of U is

$$wt_U(j) = \begin{cases} 2, \text{ if } j = 1, 3, 4, \\ 0, \text{ else.} \end{cases}$$

A (weak) set-valued shiftedtableau T is

- **increasing**, if for two distinct boxes $(x, y) \neq (a, b)$ it holds that $x \leq a$ and $y \leq b$ if and only if $\max(T_{xy}) \leq \min(T_{ab})$.
- **diagonally-unprimed**, if its main diagonal contains no (multi-)set with a primed entry.
- **standard**, if it is increasing, diagonally-unprimed and for every $i \in [|T|]$ exactly one of i and i' appears in the tableau.
- **semistandard**, if it is increasing, diagonally unprimed, each unprimed number appears in at most one box in each column of T and each primed number appears in at most one box in each row of T.

Example 15. We take a look onto the following tableaux.

	22
1	12′

	2
1	2′2′

	1
2′	12′

The first two are semistandard, while the last one is not: it is not increasing, since $\max(T_{11}) = 2' > 1 = \min(T_{12})$, the box T_{11} on the diagonal has a primed entry, the unprimed 1 appears in two boxes of the second column and the primed 2 appears in two boxes of the first row.

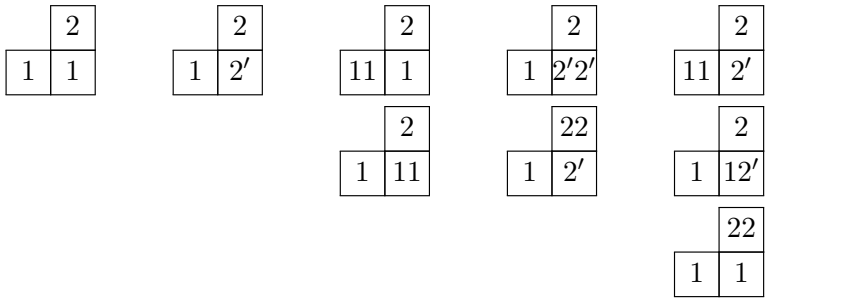
If T is a tableau, we define the monomial $x^T = \prod_i x_i^{wt_T(i)}$ and for a set of tableaux $\mathcal{T}$ we define the generating function of $\mathcal{T}$ as $\sum_{T \in \mathcal{T}} x^T$.

We define the polynomials KP_λ for a partition λ as generating function over all semistandard weak set-valued shifted tableaux of shape λ.

Example 16. We want to calculate a few factors of $\mathrm{KP}_{(2,1)}$. To get the factor of a monomial $\prod x_i^{a_i}$, we need to check the number of possible semistandard weak set-valued tableaux T of shape $(2, 1)$ with $\mathrm{wt}_T(i) = a_i$, so we try to fill the shifted Young diagram

with numbers, such that i or i' appear exactly a_i times:

$\mathrm{KP}_{(2,1)} = 1 \cdot x_1^2 x_2 + 1 \cdot x_1 x_2^2 + 2 \cdot x_1^3 x_2 + 2 \cdot x_1 x_2^3 + 3 \cdot x_1^2 x_2^2 + \ldots$



3 Insertion algorithm

In this main chapter of the thesis, we introduce the symplectic Hecke insertion, learn about its important properties, and work out its inverse. This follows primarily from the third chapter of [5]. In the final section of this chapter, we refine the insertion to obtain a semistandard version of the symplectic Hecke insertion. This refinement is based on section 4.1 of [5]. The semistandard version is the bijection we need in the next chapter to prove the main result of the thesis.

For a given strict partition λ we define the K-theoretic Schur P-function GP_λ as the generating function of all semistandard set-valued shifted tableaux of shape λ.

In Chapter 4 we see a bijective approach of a proof, that the K-theoretic Schur P-functions generating a ring with non-negative structure constants, outlined in [4]. That means, that we have to show, that $\mathrm{GP}_\lambda \cdot \mathrm{GP}_\mu = \sum_\nu e^\nu_{\lambda\mu} \mathrm{GP}_\nu$ for some non-negative integers $e^\nu_{\lambda\mu} \in \mathbb{N}^0$. The first objective is to proof the following theorem, which we accomplish at the end of section 3.2.

Theorem 3.1. Let $z \in \mathcal{F}_\infty$. There is a bijection $\mathcal{H}_{\mathrm{Sp}}(z) \longrightarrow (P, Q)$ between the set of symplectic Hecke words for z and the set of pairs of tableaux (P, Q), where P is an increasing shifted tableau with $\mathrm{row}(P) \in \mathcal{H}_{\mathrm{Sp}}(z)$ and Q is a standard shifted set-valued tableau with length $|Q| = \mathrm{len}(z)$ and the same shape as P. Additionally, if we take an FPF-involution word $w \in \mathcal{R}_{\mathrm{FPF}}(z)$, then $\mathrm{row}(P) \in \mathcal{R}_{\mathrm{FPF}}$ and Q is marked.

3.1 Symplectic Hecke insertion

In this section, we introduce the forward transition graph, which is the core of the symplectic Hecke insertion. We collect some arguments for why the insertion produces exactly one tableau for any input and prove some useful properties that we need in the next section. This follows primarily from section 3.1 and 3.2 of [5].

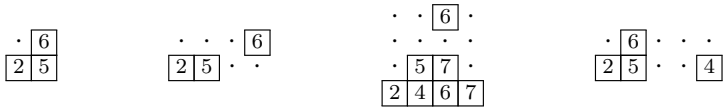
Definition 3.2. A **shifted insertion state** is a tableau that is either

- increasing, shifted and nonempty (then it is called **terminal**) or

- formed from an increasing shifted tableau with $m-2$ rows and $n-2$ columns by adding an extra box in either $\{m\} \times [n-1]$ or $[m-1] \times \{n\}$, which is called its **outer box**.

If the outer box is in $[m-1] \times \{n\}$, we call it an **outer row box**, since it is behind the rows. If it is in $\{m\} \times [n-1]$, we call it an **outer column box**, since it is above the columns.

Example 17.



The first three tableaux are shifted insertion states. The first tableau is terminal, the second one has an outer row box, while the third one has an outer column box. The last tableau is not a shifted insertion state, since the outer box for an tableau with 2 rows and 2 columns has to be either in the fourth column or in the fourth row.

We define now three families of weighted directed edges, to get a weighted directed graph. The knots for this graph are the terminal insertion states.

1. Row transitions

   Let U be a non-terminal shifted insertion state with $m-2$ rows and $n-2$ columns when its outer box is removed and let the outer box of U be in (i,n), so it is an outer row box. If the outer box is maximal in its row, let $j \in \mathbb{N}$ be minimal with $i \leq j$ and $(i,j) \notin U$, so it is the first free position in the row of the outer box.

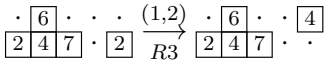
   R1: If moving the outer box of U to position (i,j) yields an increasing shifted tableau V, then there is an edge $U \xrightarrow{(i,j)} V$.

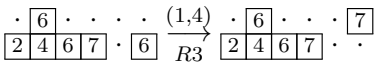
   R2: If moving the outer box of u to position (i,j) does not yield an increasing shifted tableau, then there is an edge $U \xrightarrow{(i,j)} V$, where V is formed from U by removing the outer box.

   If the outer box is not maximal in its row, there must exist a minimal $x \in \mathbb{N}$ with $(i,x) \in U$ and $U_{in} < U_{ix}$. Assume $i < x$.

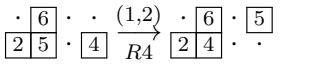
   R3: If moving the outer box to position (i,x) does not yield an increasing tableau, and $i+1 < x$ or the $i+1$-th row of U is non-empty, then there is an edge

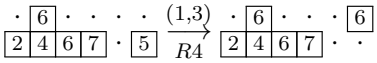
$U \xrightarrow{(i,x)} V$, where V is formed from U by moving the outer box to position $(i+1, n)$ and changing its value to U_{ix}.





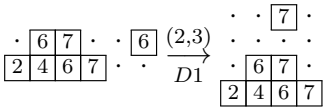
R4: If moving the outer box of U to position (i, x) yields an increasing tableau, then there is an edge $U \xrightarrow{(i,x)} V$, where V is formed from U by moving box (i, x) to $(i+1, n)$ and the outer box to (i, x).





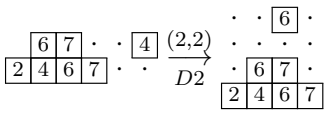
2. Diagonal transitions

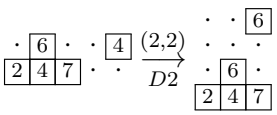
D1: If $x = i+1$ and the $i+1$-th row of U is empty, but moving the outer box to position $(i, i+1)$ does not yield an increasing tableau, then there is an edge $U \xrightarrow{(i,i+1)} V$, where V is formed from U by moving the outer box to $(m, i+1)$ and changing its value to $U_{i,i+1}$.



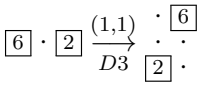
For the next three cases, suppose that the i-th row is not empty and that the entry in the outer box U_{in} is lower than the entry at the main-diagonal in this row U_{ii}. (So, we handle the case with $x = i$.)

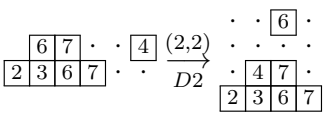
D2: If U_{in} and U_{ii} have the same parity but moving the outer box to position (i, i) does not yield an increasing tableau, then there is an edge $U \xrightarrow{(i,i)} V$, where V is formed from U by moving the outer box to $(m, i+1)$ and changing its value to U_{ii}.





D3: If U_{in} and U_{ii} have the same parity and moving the outer box to position (i, i) yields an increasing tableau, then there is an edge $U \xrightarrow{(i,i)} V$, where V is formed from U by moving box (i, i) to $(m, i+1)$ and then the outer box to (i, i).





D4: If U_{in} and U_{ii} have different parities, then there is an edge $U \xrightarrow{(i,i)} V$, where V is formed from U by moving the outer box to $(m, i+1)$ and changing its value to $U_{ii}+1$.

3. Column transitions

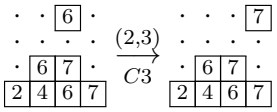
Now assume, that the outer box of U is in (m, j), so it is an outer column box. If the outer box is maximal in its column, let $i \in \mathbb{N}$ be minimal with $(i, j) \notin U$, so it is the first free box in the column of the outer box.

C1: If moving the outer box to position (i, j) yields an increasing shifted tableau V, then there is an edge $U \xrightarrow{(i,j)} V$.

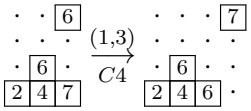
C2: If moving the outer box to position (i, j) does not yield an increasing shifted tableau V, then there is an edge $U \xrightarrow{(i,j)} V$, where V is formed from U by removing the outer box.

If the outer box is not maximal in its column, then there is a minimal $x \in \mathbb{N}$ with $(x, j) \in U$ and $U_{mj} < U_{xj}$.

C3: If moving the outer box to position (x, j) does not yield an increasing tableau, then there is an edge $U \xrightarrow{(x,j)} V$, where V is formed from U by moving the outer box to $(m, j+1)$ and changing its value to U_{xj}.



C4: If moving the outer box to position (x, j) yields an increasing tableau, then there is an edge $U \xrightarrow{(x,j)} V$, where V is formed from U by moving the box at position (x, j) to $(m, j+1)$ and the outer box to (x, j).



This ends the enumeration of the types of edges. The directed weighted graph on shifted insertion states with these edges is called the **forward transition graph**.

We make a few observations about the edges:

- All the edges start at non-terminal insertion states.
- The cases for the edges are disjunct, so there is a maximum of one outgoing edge per non-terminal insertion state.
- The cases for the edges cover all possible types of non-terminal shifted insertion states, so there is a minimum of one outgoing edge per non-terminal insertion states.

So the forward transition graph has exactly one outgoing edge at any non-terminal insertion state and the terminal insertion states are the sinks of the graph. That means, that from any insertion state, we can follow exactly one edge to another insertion state.

Assume an edge starts in U and ends in an non-terminal insertion state V. We observe the possible changes of the position of the outer box:

- If the edge is of type R3 or R4, the outer boxes of U and V has both to be outer row boxes and the one of V is just one row higher than the one of U.
- If the edge is a diagonal transition, the outer box changes from an outer row box in U to an outer column box in V, and if the outer box of U was in the k-th row it is in V in the $(k+1)$-th column.
- If the edge is of type C3 or C4 the outer box stays an outer column box and is only pushed a row rightwards.

We see here, that the paths in the forward transition graph have a special form: there can be only one diagonal transition, while there are only row transition before any diagonal transition, and the column transitions are all after a diagonal transition.

If the outer box is in the $(m-1)$-th row or the $(n-1)$-th column, it is maximal in its row and column, so the next edge has to be of type R1, R2, C1 or C2, which leads to a terminal shifted insertion state, so the path has to be finite length: if a non-terminal shifted insertion state without its outer box has $m-2$ rows and $n-2$ columns, then the path has a length of maximal $\max\{m,n\}-1$ edges.

We conclude, that for any non-terminal shifted insertion state, there is an unique path, which has to end in a terminal shifted insertion state.

With this knowledge we can define now the symplectic Hecke insertion:

Definition 3.3. Let T be an increasing shifted tableau and let $a \in \mathbb{N}$. Write $T \oplus a$ for the shifted insertion state formed by adding a to the second unoccupied box in the first row of T. A shifted insertion state is called **initial**, if its outer box is in the first row. If the directed path from $T \oplus a$ to a terminal state in the forward transition graph is

$$T \oplus a = U_0 \overset{(i_1,j_1)}{\longrightarrow} U_1 \overset{(i_2,j_2)}{\longrightarrow} U_2 \overset{(i_3,J_3)}{\longrightarrow} \dots \overset{(i_l,j_l)}{\longrightarrow} U_l,$$

then we define $T \overset{\text{Sp}}{\leftarrow} a$ to be the increasing shifted tableau U_l and call the sequence of positions $(i_1, j_1), (i_2, j_2), \ldots, (i_l, j_l)$ the **bumping path** of inserting a into T. The operation transforming (T, a) to $T \overset{\text{Sp}}{\leftarrow} a$ is called **symplectic Hecke insertion**.

For a word $w = w_1 w_2 \ldots w_n$, let

$$P_{Sp}(w) := (\ldots((\emptyset \overset{\text{Sp}}{\leftarrow} w_1) \overset{\text{Sp}}{\leftarrow} w_2) \overset{\text{Sp}}{\leftarrow} \ldots) \overset{\text{Sp}}{\leftarrow} w_n.$$

We call $P_{Sp}(w)$ the insertion tableau of w under symplectic Hecke insertion.

Example 18. We want to calculate $P_{Sp}(265425)$. The main part of the needed edges were already examples under the definitions of the types of edges.

$$\emptyset \oplus 2 = \begin{array}{|c|}\hline 2 \\ \hline \end{array} \xrightarrow[R1]{(1,1)} \begin{array}{|c|}\hline 2 \\ \hline \end{array} = \emptyset \overset{\text{Sp}}{\leftarrow} 2.$$

$$\begin{array}{|c|}\hline 2 \\ \hline \end{array} \oplus 6 = \begin{array}{|c|c|}\hline 2 & 6 \\ \hline \end{array} \xrightarrow[R1]{(1,2)} \begin{array}{|c|c|}\hline 2 & 6 \\ \hline \end{array} = \begin{array}{|c|}\hline 2 \\ \hline \end{array} \overset{\text{Sp}}{\leftarrow} 6.$$

$$\begin{array}{|c|c|}\hline 2 & 6 \\ \hline \end{array} \oplus 5 = \begin{array}{|c|c|c|}\hline 2 & 6 & 5 \\ \hline \end{array} \xrightarrow[R4]{(1,2)} \begin{array}{|c|c|}\hline 2 & 5 \\ \hline \end{array} \xrightarrow[R1]{(2,2)} \begin{array}{|c|c|}\hline 2 & 5 \\ \hline \end{array} = \begin{array}{|c|c|}\hline 2 & 6 \\ \hline \end{array} \overset{\text{Sp}}{\leftarrow} 5.$$

$$\begin{array}{|c|c|}\hline 2 & 5 \\ \hline \end{array} \oplus 4 = \begin{array}{|c|c|c|}\hline 2 & 5 & 4 \\ \hline \end{array} \xrightarrow[R4]{(1,2)} \begin{array}{|c|c|}\hline 2 & 4 \\ \hline \end{array} \xrightarrow[D4]{(2,2)} \begin{array}{|c|c|}\hline 2 & 4 \\ \hline \end{array} \xrightarrow[C1]{(1,3)} \begin{array}{|c|c|c|}\hline 2 & 4 & 7 \\ \hline \end{array} = \begin{array}{|c|c|}\hline 6 \\ \hline 2 & 5 \\ \hline \end{array} \overset{\text{Sp}}{\leftarrow} 4.$$

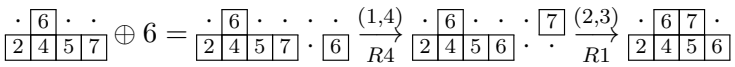
$$\begin{array}{|c|c|c|}\hline 2 & 4 & 7 \\ \hline \end{array} \oplus 2 = \begin{array}{|c|c|c|c|}\hline 2 & 4 & 7 \\ \hline \end{array} \xrightarrow[R3]{(1,2)} \begin{array}{|c|c|c|}\hline 2 & 4 & 7 \\ \hline \end{array} \xrightarrow[D2]{(2,2)} \begin{array}{|c|c|c|}\hline 2 & 4 & 7 \\ \hline \end{array} \xrightarrow[C4]{(1,3)} \begin{array}{|c|c|c|}\hline 2 & 4 & 6 \\ \hline \end{array} \xrightarrow[C1]{(1,4)} \begin{array}{|c|c|c|c|}\hline 2 & 4 & 6 & 7 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline 6 \\ \hline 2 & 4 & 7 \\ \hline \end{array} \overset{\text{Sp}}{\leftarrow} 2.$$

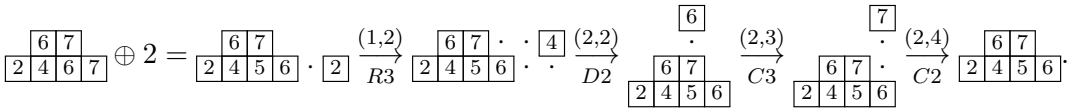
$$\begin{array}{|c|c|c|}\hline 2 & 4 & 6 \\ \hline \end{array} \oplus 5 = \begin{array}{|c|c|c|c|}\hline 2 & 4 & 5 & 7 \\ \hline \end{array} \xrightarrow[R3]{(1,4)} \begin{array}{|c|c|c|c|}\hline 2 & 4 & 5 & 7 \\ \hline \end{array} \xrightarrow[R2]{(2,3)} \begin{array}{|c|c|c|c|}\hline 2 & 4 & 5 & 7 \\ \hline \end{array}$$

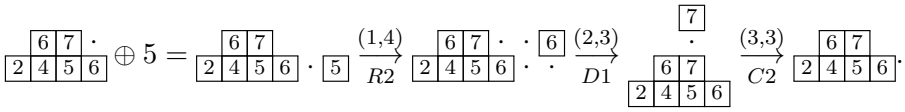
So finally, we get

$$P_{Sp}(265425) = \begin{array}{|c|}\hline 6 \\ \hline 2 & 4 & 5 & 7 \\ \hline \end{array}.$$

We can add more letters, for example:



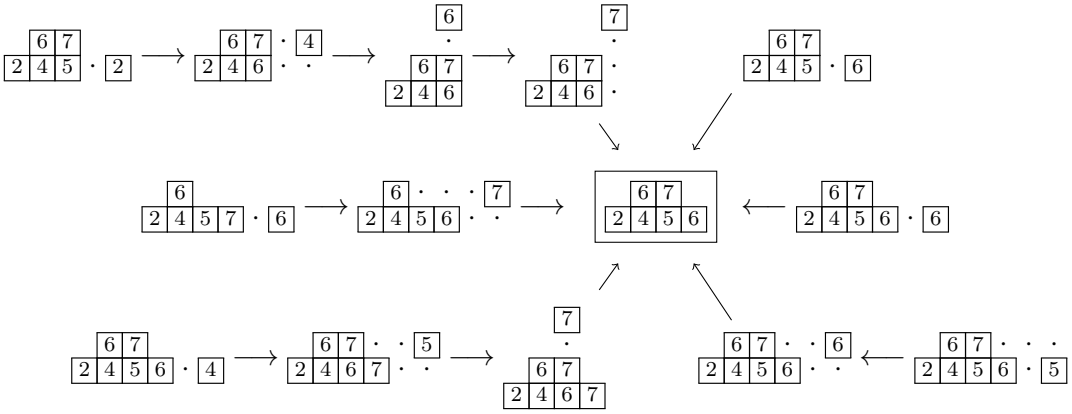




Here we can see that

$$P_{Sp}(2654256) = P_{Sp}(26542562) = P_{Sp}(26542565) = P_{Sp}(265425625).$$

Example 19. We see later how we can use the insertion for a bijection. Till now, there are many paths in the forward transition graph which lead to the same tableau. We look onto a subgraph of the forward transition graph for an example:



For the remaining of this section, we look onto some properties of this insertion.

Definition 3.4. Let T be a shifted insertion state with outer box (i, j). Assume T with its outer box removed has $m-2$ rows and $n-2$ columns, and set $T_{xy} := \infty$ for all positions $(x, y) \notin T$. We say that T is **weakly admissible** if one of the following conditions holds:

- When $j = n$ (t.m. when the outer box of T is an outer row box) and either $i = 1$ or there exists a column $x \geq i$ with $T_{i-1,x} \leq T_{in} < T_{ix}$, and if $T_{i-1,i} = T_{in}$, then $(i, i) \in T$.
- When $i = m$ (t.m. when the outer box of T is an outer column box) and either $T_{mj} = T_{j-1,j}$ or there exists a row $x < j$ with $T_{x,j-1} \leq T_{mj} < T_{xj}$.

- Additionally, we also say that any terminal shifted insertion state is weakly admissible.

Example 20. Every initial shifted insertion state is weakly admissible, and a weakly admissible shifted insertion state cannot have his outer column box in the first column.

The first tableau, T, is weakly admissible, since the outer box $T_{24} = 6 < T_{22} = \infty$ and $T_{12} = 5 < 6 = T_{24}$.

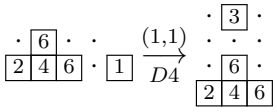
The second tableau, S, is weakly admissible, since $S_{11} \leq S_{42} = S_{12}$.

The third tableau, U, is not weakly admissible, since although the second row supplies $U_{23} = U_{43}$, which fulfills the second condition, the tableau without the outer box is not weakly increasing.

The fourth tableau is not weakly increasing, since for every column after the first the value of the outer box, 5, is not smaller or equal than the first and smaller than the second entry.

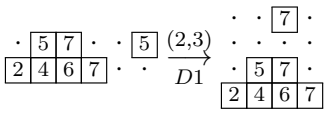
Proposition 3.5. If $U \to V$ is an edge in the forward transition graph, then V is weakly admissible.

Example 21.



We already saw in the example before, that the second tableau is weakly admissible. Since the first tableau is initial, it is also weakly admissible.

But also if the first tableau is a shifted insertion state, which is not weakly admissible, the tableau at the end of an edge in the transition graph is:



The first tableau is not weakly admissible, as seen in the example before. But the second one, let us name it T, is, since the outer box has the value $T_{43} = 7 = T_{23}$.

Lemma 3.6. Let $U \to V$ be an edge in the forward transition graph between weakly admissible shifted insertion states.

If it is a row transition, then $\mathrm{row}(U) \overset{\mathrm{K}}{\approx} \mathrm{row}(V)$ and if additionally $\mathrm{row}(U)$ is reduced then $\mathrm{row}(U) \overset{\mathrm{K}}{\sim} \mathrm{row}(V)$.

If it is a column transition, then $\mathrm{col}(U) \overset{\mathrm{K}}{\approx} \mathrm{col}(V)$ and if additionally $\mathrm{col}(U)$ is reduced, then $\mathrm{col}(U) \overset{\mathrm{K}}{\sim} \mathrm{col}(V)$.

Proof. We only proof the first part of the first statement.

If the edge $U \to V$ is of type (R1), the row reading words of U and V are the same. If the edge is of type (R4), let us call the entries of the row with the outer box $a_1 < a_2 < \cdots < a_k$, while b is the entry of the outer box. We need to have $a_{x-1} < b < a_x$ for $1 \leq x \leq k$. In this case, we can see that

$$\begin{aligned}\mathrm{row}(U) &= \ldots a_1 \ldots a_{x-1} a_x a_{x+1} \ldots a_k b \\ &\overset{\mathrm{K}}{\sim} \ldots a_1 \ldots a_{x-1} a_x b a_{x+1} \ldots a_k \\ &\overset{\mathrm{K}}{\sim} \ldots a_x a_1 \ldots a_{x-1} b a_{x+1} \ldots a_k = \mathrm{row}(V).\end{aligned}$$

Assume that the outer box of U appears in the first row. If it is of type (R2), then the value of the outer box and the value of the last box in its row without the outer box has to be equal, so the row reading words of U and V has to be connected by $\overset{\mathrm{K}}{\approx}$, because of $X \overset{\mathrm{K}}{\approx} XX$ for every $X \in \mathbb{N}$. Assume the first row of U has k boxes with values $a_1 < a_2 < \cdots < a_k$ and the value of the outer box is b. If the edge $U \to V$ is of type (R3), we need to have another box in the first row, that has the same value as the outer box, let us say $a_{x-1} = b < a_x$, for $1 \leq x \leq k$. Since $b = a_{x-1} < a_l < a_{l+1}$ for every $l \geq x$, we know that $a_l a_{l+1} a_{x-1} \overset{\mathrm{K}}{\approx} a_l a_{x-1} a_{l-1}$. Similiary, since $b = a_{x-1} > a_m < a_{m-1}$ for every $m \leq x$, we know that $a_{x-1} a_{m-1} a_m \overset{\mathrm{K}}{\approx} a_{m-1} a_{x-1} a_m$. So

$$\begin{aligned}\mathrm{row}(U) &= \ldots a_1 \ldots a_{x-1} a_x a_{x+1} \ldots a_k a_{x-1} \\ &\overset{\mathrm{K}}{\sim} \ldots a_1 \ldots a_{x-1} a_x a_{x-1} a_{x+1} \ldots a_k \\ &\overset{\mathrm{K}}{\sim} \ldots a_1 \ldots a_x a_{x-1} a_x a_{x+1} \ldots a_k \\ &\overset{\mathrm{K}}{\sim} a_x a_1 \ldots a_{x-1} a_x a_{x+1} \ldots a_k = \mathrm{row}(V).\end{aligned}$$

Now assume, that the outer box of U appears in any row above the first, so two rows of U have the form

	a_2	$\ldots$	a_k	$\cdot$	$\cdot$	b
c_1	c_2	$\ldots$	$\ldots$	c_l	$\cdot$	$\cdot$

.

If $k = 0$, then $c_2 \neq b$, since U is weakly admissible. Then the edge $U \to V$ has to be of type (R1).

Assume $U \to V$ is of type (R2). Then $a_k = b$ or $c_{k+1} = b$. In the first case, we can show as above that $\mathrm{row}(U) \overset{\mathrm{K}}{\approx} \mathrm{row}(V)$. So assume $c_{k+1} = b$. Since $c_1 < \cdots < c_k < a_k < b$, we

see that

$$\begin{aligned}
\mathrm{row}(U) &= \dots a_k b c_1 c_2 \dots c_k b c_{k+2} \dots \\
&\overset{\mathrm{K}}{\sim} \dots a_k c_1 c_2 \dots c_{k-1} b c_k b c_{k+2} \dots \\
&\overset{\mathrm{K}}{\sim} \dots c_1 c_2 \dots c_{k-1} a_k b c_k b c_{k+2} \dots \\
&\overset{\mathrm{K}}{\sim} \dots c_1 c_2 \dots c_{k-1} a_k c_k b c_k c_{k+2} \dots \\
&\overset{\mathrm{K}}{\sim} \dots c_1 c_2 \dots c_{k-1} a_k b c_k c_k c_{k+2} \dots \\
&\overset{\mathrm{K}}{\approx} \dots c_1 c_2 \dots c_{k-1} a_k b c_k c_{k+2} \dots \\
&\overset{\mathrm{K}}{\sim} \dots a_k c_1 c_2 \dots c_{k-1} b c_k c_{k+2} \cdots = \mathrm{row}(V).
\end{aligned}$$

Finally, we assume $U \to V$ is of type (R3). Define x as in the definition of (R3). It follows that $a_{x-1} = b$ or $c_x = b$, since otherwise we would have an edge of type (R4). If $a_{x-1} = b$ we can follow $\mathrm{row}(U) \overset{\mathrm{K}}{\sim} \mathrm{row}(V)$ as above. So assume $c_x = b$ and $a_{x-1} < b < a_x$. It follows that

$$\begin{aligned}
\mathrm{row}(U) &= a_2 \dots a_k b c_1 \dots c_{x-1} b \dots \\
&\overset{\mathrm{K}}{\sim} a_2 \dots a_{x-1} a_x b a_{x+1} \dots a_k c_1 \dots c_{x-1} b \\
&\overset{\mathrm{K}}{\approx} a_2 \dots a_{x-1} a_{x-1} a_x b a_{x+1} \dots a_k c_1 \dots c_{x-1} b \\
&\overset{\mathrm{K}}{\sim} a_2 \dots a_{x-1} a_x a_{x-1} b a_{x+1} \dots a_k c_1 \dots c_{x-1} b \\
&\overset{\mathrm{K}}{\sim} a_2 \dots a_x a_{x-1} a_x b a_{x+1} \dots a_k c_1 \dots c_{x-1} b \\
&\overset{\mathrm{K}}{\sim} a_2 \dots a_x a_{x-1} a_x a_{x+1} \dots a_k b c_1 \dots c_{x-1} b \\
&\overset{\mathrm{K}}{\sim} a_x a_2 \dots a_x a_{x-1} a_{x+1} \dots a_k b c_1 \dots c_{x-1} b.
\end{aligned}$$

And as before, where the edge was of type (R2), we can eliminate the first b, so $row(U) \overset{\mathrm{K}}{\approx} a_x a_2 \dots a_k c_1 \dots c_{x-1} b = a_x a_2 \dots a_k c_1 \dots c_{x-1} b \cdots = \mathrm{row}(V)$. □

Lemma 3.7. Suppose $U \to V$ is a diagonal transition between weakly admissible shifted insertion states. Assume (i, n) is the outer box of U, so that $(i, i) \in U$.

1. If $U \to V$ is of type (D1) and $U_{ii} \equiv U_{i,i+1} \mod 2$, then $\mathrm{row}(U) \overset{\mathrm{Sp}}{\approx} \mathrm{col}(V)$.
2. If $U \to V$ is of type (D2), (D3), or (D4), all entries on the main diagonal of U have the same parity, and either $U_{in} \equiv U_{ii} \mod 2$ or $U_{in} = U_{ii} - 1$, then $\mathrm{row}(U) \overset{\mathrm{K}}{\sim} \mathrm{col}(V)$.
3. If $\mathrm{row}(U)$ is a symplectic Hecke word, then $\mathrm{row}(U) \overset{\mathrm{Sp}}{\approx} \mathrm{col}(V)$, and if $\mathrm{row}(U)$ is a symplectic Hecke word that is also a reduced word, then $\mathrm{row}(U) \overset{\mathrm{Sp}}{\sim} \mathrm{col}(U)$.

We summarise the last two lemmas for our main theorem of this section:

Theorem 3.8. Suppose T is an increasing shifted tableau and $a \in \mathbb{N}$ is such that $\mathrm{row}(T)a$ is a symplectic Hecke word. The following properties then hold:

1. The tableau $T \overset{\mathrm{Sp}}{\leftarrow} a$ is increasing and shifted with $\mathrm{row}(T \overset{\mathrm{Sp}}{\leftarrow} a) \overset{\mathrm{Sp}}{\approx} \mathrm{row}(T)a$.

2. If $\mathrm{row}(T)a$ is an FPF-involution word, then $\mathrm{row}(T \overset{\mathrm{Sp}}{\leftarrow} a) \overset{\mathrm{Sp}}{\sim} \mathrm{row}(T)a$.

Since the insertion tableau $P_{Sp}(w)$ of w under symplectic Hecke insertion is only a recursive insertion of all the letters of w, the theorem implies the following lemma.

Lemma 3.9. If w is a symplectic Hecke word, then $w \overset{\mathrm{Sp}}{\approx} \mathrm{row}(P_{Sp}(w))$. If w is an FPF-involution word, then $w \overset{\mathrm{Sp}}{\sim} \mathrm{row}(P_{Sp}(w))$.

Example 22. We saw in Example 18 that

$$P(26542) = \begin{array}{|c|c|c|c|}\hline \multicolumn{1}{c}{} & \multicolumn{1}{c}{} & \multicolumn{1}{|c|}{6} & \multicolumn{1}{c}{} \\ \hline 2 & 4 & 6 & 7 \\ \hline \end{array} := T.$$

Furthermore, we saw in Example 5 that $\mathrm{row}(T) = 62467 \overset{\mathrm{Sp}}{\approx} 26542$.

3.2 Inverse insertion

In the first part of this section we introduce a stronger version of admissibility, which sum up the insertion states, which are able to appear if we compute the insertion tableau of a symplectic Hecke word. For them we define the inverse edges of the forward insertion graph. After that, we define the recording tableau and proof Theorem 3.1. This mainly follows section 3.3 and 3.4 of [5].

Definition 3.10. Let T be a shifted insertion state. We define

$$\mathrm{word}(T) := \begin{cases} \mathrm{col}(T) & , \text{ if } T \text{ has an outer column box,} \\ \mathrm{row}(T) & , \text{ otherwise.} \end{cases}$$

Definition 3.11. Let T be a shifted insertion state. Assume T without (i, j) has $m-2$ rows and $n-2$ columns. We say, that T is **admissible**, if T is weakly admissible, $\mathrm{word}(T)$ is a symplectic Hecke word and the following condition holds:

1. if (i, j) is an outer column box, it holds that if $T_{mj} = T_{j-1,j}$ then $(j, j) \notin T$ or T_{mj} is odd, and if $T_{mj} = T_{x,j-1}$ then $x \geq 2$.

Example 23. Every terminal shifted insertion state with a row (or equivalently column) reading word, which is a symplectic Hecke word, is admissible, since it is weakly admissible and has no outer box.

$$\begin{array}{|c|c|}\hline \cdot & 6 \\ \hline 2 & 5 \\ \hline \end{array} \cdot \begin{array}{|c|}\hline 3 \\ \hline \cdot \end{array} \qquad \begin{array}{|c|c|}\hline \cdot & \cdot \\ \hline 2 & 5 \\ \hline \end{array} \cdot \begin{array}{|c|}\hline 6 \\ \hline \cdot \end{array} \qquad \begin{array}{|c|c|c|c|}\hline \cdot & \cdot & 7 & \cdot \\ \hline \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \cdot & 8 & \cdot \\ \hline \cdot & 6 & 7 & \cdot \\ \hline 2 & 4 & 6 & 7 \\ \hline \end{array} \qquad \begin{array}{|c|}\hline \cdot \\ \hline \cdot \\ \hline 2 \\ \hline \end{array} \cdot \begin{array}{|c|}\hline 4 \\ \hline \cdot \\ \hline \end{array}$$

The first insertion state is not admissible, since the row reading word is not a symplectic Hecke word, since $((67)\theta(67))(3) = 4$ and $((67)\theta(67))(4) = 3$, so $\theta \odot 6 \odot 3 = 0$.

The second insertion state, T, is admissible: it is weakly admissible (as proven in Example 20), the word $625 \overset{\text{Br}}{\equiv} 265$, which is a symplectic Hecke word as shown in Example 4 and $T_{24} = 6 \neq 5 = T_{12}$.

The third insertion state, S, is not admissible, since the outer box $S_{53} = 7 = S_{23}$ but $(3, 3)$ is occupied by the tableau.

The fourth insertion state, U, is admissible, since its column reading word 244 is not $\overset{\text{Sp}}{\equiv}$-equivalent to a word which starts with an odd letter (see Lemma 2.2), it is weakly admissible, since $U_{12} = U_{32}$, and although the outer box is even, $(2, 2)$ is not occupied by U.

Proposition 3.12. Suppose T is an admissible shifted insertion state. Assume that T has r rows with its outer box removed (if one exists). The diagonal entries T_{kk} for $k \in [r]$ are then all even.

Proof. Let $k \in [r]$. Since T is admissible, we know that $\text{word}(T) \in \mathcal{H}_{\text{Sp}}(z)$ for a $z \in \mathcal{F}_\infty$. We want to show that there is a word in the $\overset{\text{Sp}}{\equiv}$-equivalence class of $\text{word}(T)$, which begins with T_{kk}. This implies that T_{kk} has to be even by Lemma 2.2.

First assume there is either no outer box or the outer box of T is in the k-th row or beneath it. Since T is increasing, we know that $T_{kk} < T_{k,k+1} < T_{xy}$ for every $x < k, y < x$. So $|T_{kk} - T_{xy}| > 1$ for every $x < k, y < x$, so we can swap T_{kk} with every element that appears before it in $\text{row}(T)$:

$$\begin{aligned}
\text{row}(T) &= T_{rr}T_{r,r+1}\dots T_{r,s_r}\dots\dots T_{k+1,s_{k+1}-1}T_{k+1,s_{k+1}}T_{kk}\dots \\
&\overset{\text{Br}}{\equiv} T_{rr}T_{r,r+1}\dots T_{r,s_r}\dots\dots T_{k+1,s_{k+1}-1}T_{kk}T_{k+1,s_{k+1}} \\
&\dots \\
&\overset{\text{Br}}{\equiv} T_{kk}T_{rr}T_{r,r+1}\dots T_{r,s_r}\dots\dots T_{k+1,s_{k+1}-1}T_{k+1,s_{k+1}}.
\end{aligned}$$

Now assume T has an outer row box at position (i, j) in a row higher than k. Since T is admissible, there has to be a column $x \geq i$ with $T_{i-1,x} \leq T_{ij}$, so it has to be greater or equal to $T_{k,k+1}$. If the value in the outer box is greater than $T_{k,k+1}$, we can also swap T_{kk} with T_{ij} in $\text{row}(T)$. If the value in the outer box is equal to $T_{k,k+1}$, it has to be in row $k+1$, since otherwise $T_{i-1,x} < T_{k,k+1}$, but $i - 1 > k$ and T would not be increasing. So we can first see that

$$\begin{aligned}
\text{row}(T) &= T_{rr}T_{r,r+1}\dots T_{r,s_r}\dots\dots T_{k+1,s_{k+1}-1}T_{k+1,s_{k+1}}T_{i,j}T_{kk}T_{k,k+1}\dots \\
&= T_{rr}T_{r,r+1}\dots T_{r,s_r}\dots\dots T_{k+1,s_{k+1}-1}T_{k+1,s_{k+1}}T_{k,k+1}T_{kk}T_{k,k+1}\dots \\
&\overset{\text{Br}}{\equiv} T_{rr}T_{r,r+1}\dots T_{r,s_r}\dots\dots T_{k+1,s_{k+1}-1}T_{k+1,s_{k+1}}T_{kk}T_{k,k+1}T_{kk}\dots
\end{aligned}$$

and then make T_{kk} be the first letter of the word as before.

Now assume T has an outer column box at position (i, j). For all entries T_{xy} except the outer box, which appears in front of T_{kk} in $\text{word}(T) = \text{col}(T)$, it holds that $|T_{kk} - T_{xy}| > 1$, so we could swap them with T_{kk}.

If the outer box is in a column right of the k-th, it appears in $\text{word}(T)$ behind T_{kk}.

If the outer box is in the k-th column or in a column left from it, we can see that we have either $T_{ij} = T_{j-1,j} \leq T_{k-1,k}$ or there exists a row $x < j$ with $T_{ij} < T_{xj} \leq T_{k-1,k}$, since T is weakly admissible (in both cases $=$ is only possible, if $j = k$).

If $T_{ij} < T_{k-1,k}$, it holds that $|T_{kk} - T_{ij}| > 1$. If $T_{ij} = T_{k-1,k}$, the outer box has to be in the k-th column, so we know that

$$\begin{aligned}\text{col}(T) &= T_{11}T_{22}T_{21}\ldots\ldots T_{1,k-1}T_{ij}T_{kk}T_{k-1,k}\ldots \\ &= T_{11}T_{22}T_{21}\ldots\ldots T_{1,k-1}T_{k-1,k}T_{kk}T_{k-1,k}\ldots \\ &= T_{11}T_{22}T_{21}\ldots\ldots T_{1,k-1}T_{kk}T_{k-1,k}T_{kk}\ldots.\end{aligned}$$

In every case, in which we have an outer column box, we can swap now every letter that is in front of T_{kk} with T_{kk}. □

Proposition 3.13. Suppose $U \to V$ is a forward transition between shifted insertion states. If U is admissible, then $\text{word}(U) \overset{\text{Sp}}{\approx} \text{word}(V)$ and V is admissible.

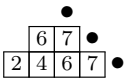
This stresses the importance of admissible tableaux. We saw earlier in 3.5 that if $U \longrightarrow V$ is an edge in the forward transition graph, then V has to be weakly admissible. Proposition 3.13 tells us furthermore, that if we start with an admissible tableau U and an edge $U \longrightarrow V$, then V even has to be admissible. This implies that the set of admissible tableaux spanns a subgraph in the forward transition graph. We define the inverse of the symplectic Hecke insertion only on this subgraph and we see, that this is exactly what we want. Especially, we can already see that we are on the right way for Theorem 3.1, since P has to be an admissible tableau, because it has to be terminal and its row reading word $\text{row}(P) \in \mathcal{H}_{\text{Sp}}(z)$ is a symplectic Hecke word.

Before we start to define the inverse edges, we need to talk about some important positions of a tableau.

Definition 3.14. Let T be a shifted tableau. A position (i, j) for $i, j \in \mathbb{N}$ is an **inner corner**, if it is the last occupied box in its row and at highest occupied position in its column. A position (i, j) for $i, j \in \mathbb{N}$ is an **outer corner**, if it is not in T, either it is on the diagonal or it is on the right of an box in T, and either it is in the first row or it is on the top of an box in T.

So an inner corner is a position, where you can remove a box and get a shifted tableau, an outer corner is a position, where you can add a box and get a shifted tableau.

Example 24. We take our tableau from Example 18 again and mark the outer corners with bullets, while the inner corners are the boxes with the value 7:



Lemma 3.15. Suppose $U \xrightarrow{(i,j)} V$ is a forward tansition between shifted insertion states, where U is admissible and V is terminal. Then $U \longrightarrow V$ is a row or column transition and (i, j) is an inner or outer corner of V. In addition, the following properties hold:

1. If $U \longrightarrow V$ is a row transition and (i, j) is an outer corner of V, then $i < j$.
2. If $U \longrightarrow V$ is a column transition and (i, j) is an inner corner of V, then $i < j$.
3. If $U \longrightarrow V$ is a column transition and (i, j) is an outer corner of V, then $i > 1$.

Proof. 1. Since V is terminal, the edge $U \longrightarrow V$ can only be of type (R1), (R2), (C1), or (C2). Assume it is a row transition and (i, j) is an outer corner of V with $i = j$. If the edge would be of type (R1), the last position in the bumping-path has to be an inner corner, so the edge has be of type (R2). This means, that the value of the outer box is either equal to $U_{i-1,j}$ or equal to $U_{i,j-1}$. Since $(i - 1, i)$ is not a valid position in a shifted tableau, we need to have the value of the outer box equal to $U_{i,i-1}$. But U is admissible which leads to $(i, j) = (i, i) \in T$, so it could not be an outer corner. So we contradicted $i = j$.

2. Now let us assume $U \longrightarrow V$ is a column transition, (i, j) is an inner corner of V, and $i = j$. If the edge is of type (C2), (i, j) has not to be occupied by V, so the edge has to be of type (C1). Then the outer box of U has to be maximal in its column. But since U is admissible, the outer box has to be equal to $U_{j-1,j}$ or smaller than any U_{xj} for an $x < j$. So we contradicted $i = j$.

3. Let us assume $U \longrightarrow V$ is a column transition, (i, j) is an outer corner of V, and $i = 1$. Again, since $(1, j)$ is an outer corner, the type of the edge can only be (C2). But then $U_{1,j-1}$ has to be equal to the value of the outer box, so U cannot be admissible. So we contradicted $i = 1$.

□

We start now to define a graph on the set of admissible tableaux. As earlier at the definition of the edges of the forward transition graph, we define different types of edges. For every type of edge of the form $V \rightsquigarrow U$ with an admissible V we add three things: first we see an example, directly below we prove that U is admissible, so the edges are well-defined, and we look for the arguments that $U \longrightarrow V$ is an edge in the forward transition graph of one specific type. A short remark at this point: since the outer box of U is either in the $i - 1$ row or column, the conditions for (weakly) admissible tableaux move in the indexes according to that.

We start with the edges on terminal tableaux V:

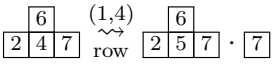
iR1: For each inner corner (i, j) of V, there is an edge $V \rightsquigarrow U$ where U is formed from V by moving box (i, j) to an outer position in row i.

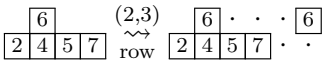
$\boxed{2} \overset{(1,1)}{\underset{\text{row}}{\rightsquigarrow}} \cdot \boxed{2}$

$\begin{array}{|c|c|}\hline & 6 \\ \hline 2 & 5 \\ \hline \end{array} \cdot \cdot \overset{(2,2)}{\underset{\text{row}}{\rightsquigarrow}} \cdot \cdot \cdot \begin{array}{|c|}\hline 6 \\ \hline \end{array}$
$\begin{array}{|c|c|}\hline 2 & 5 \\ \hline \end{array} \cdot \cdot$

We can see that U is also admissible: since $U_{i-1,j} = V_{i-1,j} < V_{ij} < U_{i,j+1} = \infty$ and V_{ij} is the value of the outer box of U, U is weakly admissible; since we only move one box sideways, it holds that $\mathrm{row}(U) = \mathrm{row}(V)$, so $\mathrm{row}(U)$ is a symplectic Hecke word; and $U_{i-1,i} = V_{i-1,i} < V_{ii} \leq V_{ij}$, such that the outer box of U is never equal to $U_{i-1,i}$. Additionally, we can see that $U \xrightarrow{(i,j)} V$ is a row transition of type R1.

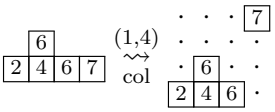
iR2: For each outer corner (i, j) of V with $i < j$, there is an edge $V \rightsquigarrow U$ where U is formed from V by adding an outer box in row i whose value is whichever of $V_{i-1,j}$ or $V_{i,j-1}$ is defined and larger.

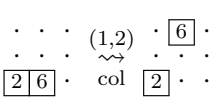




In this case $U \xrightarrow{(i,j)} V$ is a row transition of type R2, so we can conclude that $\mathrm{row}(U) \overset{\mathrm{K}}{\approx} \mathrm{row}(V)$ from Proposition 3.8. Additionally, since the outer box of U has the value $u := \max(V_{i-1,j}, V_{i,j-1})$, it holds that $U_{i-1,j} = V_{i-1,j} \leq u < U_{i,j} = \infty$ and if $u = U_{i-1,i}$, then $j = i$, which contradicts $i < j$, so U is also admissible.

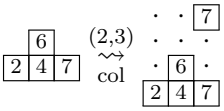
iC1: For each inner corner (i, j) of V with $i < j$, there is an edge $V \rightsquigarrow U$ where U is formed from V by moving box (i, j) to an outer position in column j.

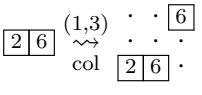




It is clear that U is also admissible and that $U \xrightarrow{(i,j)} V$ is a column transition of type (C1).

iC2: For each outer corner (i, j) of V with $i > 1$, there is an edge $V \rightsquigarrow U$ where U is formed from V by adding an outer box in column j whose value is whichever of $V_{i-1,j}$ or $V_{i,j-1}$ is defined and larger.





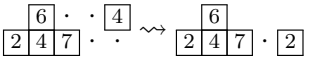
In this case $U \xrightarrow{(i,j)} V$ is a column transition of type C2, so we have $\mathrm{col}(U) \overset{\mathrm{K}}{\approx} \mathrm{col}(V)$. It follows that U is also admissible.

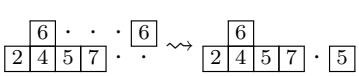
To distinguish between these edges we write $V \overset{(i,j)}{\underset{\mathrm{row}}{\rightsquigarrow}} U$ to indicate the inverse transitions of type iR1 and iR2 or $V \overset{(i,j)}{\underset{\mathrm{col}}{\rightsquigarrow}} U$ for iC1 and iC2, while in both cases (i, j) is the inner or outer corner of the terminal tableau V.

Now we look onto admissible tableaux which are neither initial nor terminal, so they have an outer box outside of the first row. Assume V is an admissible tableau with an outer box, but with its outer box removed it has $m - 2$ rows and $n - 2$ columns.

Assume V has an outer row box in (i, n). Since V is weakly admissible, there exists a column $x \geq i$ with $V_{i-1,x} \leq V_{in} < V_{ix}$. We choose this x maximal, so $V_{in} < V_{i-1,x+1}$ also holds (else $V_{i-1,x+1} \leq V_{in} < V_{i,x+1}$ and x is not maximal).

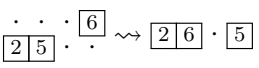
iR3: If $V_{i-1,x} = V_{in}$, then there is an edge $V \rightsquigarrow U$ where U is formed from V by moving box (i, n) to $(i-1, n)$ and changing its value to whichever of $V_{i-1,x-1}$ or $V_{i-2,x}$ is defined and larger.

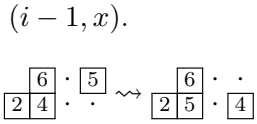




We can see, that U is weakly admissible, since $U_{i-1,n} = \max(V_{i-1,x-1}, V_{i-2,x})$, and so it holds that $U_{i-2,x} = V_{i-2,x} \leq U_{i-1,n} < V_{i-1,x} = U_{i-1,x}$. To show that $U \overset{(i-1,x)}{\longrightarrow} V$ is a row transition of type R3 we have to ensure that the i-th row of U is not empty or $i < x$ for the minimal column x for which $U_{i-1,n} < U_{i-1,x}$. If $x = i$, then $V_{in} = V_{i-1,i}$ and since V is admissible, it holds that $(i, i) \in V$ and so $(i, i) \in U$. If $x > i$, then $U_{i-1,n} < U_{i-1,x}$ and x is minimal with this property. So $U \overset{(i-1,x)}{\longrightarrow} V$ is a row transition of type (R3). Lemma 3.6 implies that $\mathrm{row}(U)$ is a symplectic Hecke word, because $\mathrm{row}(V)$ is one and finally $(i-1, x) \in V$ implies $(i-1, i) \in V$ and $(i-1, i-1) \in U$, so U is admissible.

iR4: If $V_{i-1,x} < V_{in}$, then there is an edge $V \rightsquigarrow U$ where U is formed from V by moving box $(i-1, x)$ to $(i-1, n)$ and then box (i, n) to $(i-1, x)$.

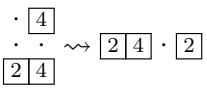


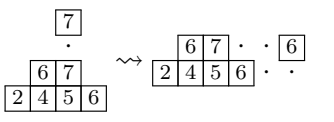


Since the outer box of U fits into position $(i-1, x)$ of U to preserve an increasing tableau, we get $U_{i-2,x} = V_{i-2,x} < V_{i-1,x} = U_{i-1,n} < V_{in} = U_{i-1,x}$, so U is weakly admissible. In this case $U \overset{(i-1,x)}{\longrightarrow} V$ is a row transition of type R4, so it follows again by Lemma 3.6 that $\mathrm{row}(U) \overset{\mathrm{K}}{\approx} \mathrm{row}(V)$ and since V is admissible, both words are symplectic Hecke words. Finally, we see that $U_{i-1,n} = V_{i-1,x} > V_{i-2,i-1} = U_{i-2,i-1}$, since $x \geq i$ and V is increasing, so U is admissible.

Now we assume the outer box of V is an outer column box and $V_{j-1,j-1} \leq V_{mj}$. Since V is weakly admissible, it holds that $V_{mj} = V_{j-1,j}$ or it exists a $x < j$ with $V_{x,j-1} \leq V_{mj} < V_{x,j} \leq V_{j-1,j}$, so in both cases $V_{mj} \leq V_{j-1,j}$. The edge $V \rightsquigarrow U$ is then one of the following types:

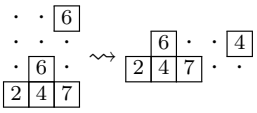
iD1: Suppose V_{mj} is even and $V_{j-1,j-1} < V_{mj} = V_{j-1,j}$, so that $(j, j) \notin V$. There is an edge $V \rightsquigarrow U$ where U is formed from V by moving box (m, j) to $(j-1, n)$ and changing its value to whichever of $V_{j-1,j-1}$ or $V_{j-2,j}$ is defined and larger.

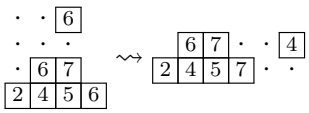




Since either $j-1=1$ or $U_{j-2,j}=V_{j-2,j}\leq U_{j-1,n}<V_{j-1,j}=U_{j-1,j}$, U is weakly admissible. We see that $U\longrightarrow V$ is a diagonal transition of type D1. We know from Lemma 3.12, that $V_{j-1,j-1}=U_{j-1,j-1}$ is even and since $V_{mj}=V_{j-1,j}$ is even, $U_{j-1,j}$ is even too. Then the first part of Lemma 3.7 tells us, that $\mathrm{row}(U)\overset{\mathrm{Sp}}{\approx}\mathrm{col}(V)$, so $\mathrm{row}(U)$ is a symplectic Hecke word. Since V is increasing, $U_{j-2,j-1}=V_{j-2,j-2}$ has to be smaller than $V_{j-1,j-1}$ and $V_{j-2,j}$, so in this case $U_{j-1,n}\neq U_{j-2,j-1}$, so U is admissible.

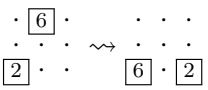
iD2: Suppose $V_{j-1,j-1}=V_{mj}$. Since V is admissible, it has to be that $j-1\geq 2$. If $V_{j-2,j-1}$ is even, then there is an edge $V\rightsquigarrow U$ where U is formed from V by moving box (m,j) to $(j-1,n)$ and changing its value to $V_{j-2,j-1}$.





First, we see again that by $U_{j-2,j-1}=U_{j-1,n}<U_{j-1,j-1}$, U is weakly admissible. Since Proposition 3.12 tells us that $V_{j-1,j-1}$ is even, $U\longrightarrow V$ is a diagonal transition of type D2. Now the second part of Lemma 3.7 shows, that $\mathrm{row}(U)$ is a symplectic Hecke word, since Lemma 3.12. So U is admissible again, since $(j-1,j-1)\in U$.

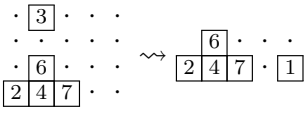
iD3: If $V_{j-1,j-1}<V_{mj}<V_{j-1,j}$ and V_{mj} is even, then there is an edge $V\rightsquigarrow U$ where U is formed from V by moving box $(j-1,j-1)$ to $(j-1,n)$ and then box (m,j) to $(j-1,j-1)$.

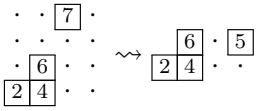


$\mathcal{E}$

Since moving box $(j-1,n)$ to $(j-1,j-1)$ in U get us an increasing tableau (it is V without its outer box), U is weakly admissible. Additionally, by Proposition 3.12, $V_{j-1,j-1}$ is even, so $U\longrightarrow V$ is a diagonal transition of type D3. All the other entries on the main diagonal are also even by 3.12, and so 3.7 implies $\mathrm{col}(V)\overset{\mathrm{Sp}}{\sim}\mathrm{row}(U)$ and $\mathrm{row}(U)$ is a symplectic Hecke word. Since $U_{j-1,n}=V_{j-1,j-1}>V_{j-2,j-1}=U_{j-2,j-1}$, if these entries exist, it holds that U is admissible.

iD4: If $V_{j-1,j-1}<V_{mj}$ and V_{mj} is odd, then there is an edge $V\rightsquigarrow U$ where U is formed from V by moving box (m,j) to $(j-1,n)$ and changing its value to $V_{j-1,j-1}-1$.

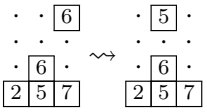




$V_{j-1,j-1}=U_{j-1,j-1}$ has to be even because of 3.12, so $U_{j-1,n}$ is odd. Then $U\longrightarrow V$ is a diagonal transition of type D4. Since the entries on the main diagonal of V have not changed and Lemma 3.12 shows, that the ones of V are even, the entries

on the main diagonal of U are also even. Then $\mathrm{col}(V) \overset{\mathrm{Sp}}{\sim} \mathrm{row}(U)$ by Lemma 3.7, so $\mathrm{row}(U)$ is a symplectic Hecke word. Since $(j-1, j-1) \in U$, U is admissible.

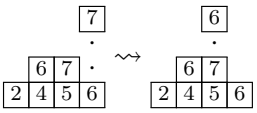
iC3a: Suppose $V_{j-1,j-1} = V_{mj}$, so that $j - 1 \geq 2$, since V is admissible. If $V_{j-2,j-1}$ is odd, then there is an edge $V \rightsquigarrow U$ where U is formed from V by moving box (m, j) to $(m, j-1)$ and changing its value to $V_{j-2,j-1}$.



In this case $U \longrightarrow V$ is a column transition of type C3, so $\mathrm{col}(U) \overset{\mathrm{K}}{\sim} \mathrm{col}(V)$ has to hold by Lemma 3.6. The tableau U has to be weakly admissible, since $U_{m,j-1} = U_{j-2,j-1}$. Since $U_{m,j-1}$ is odd and since $U_{m,j-1} = U_{j-2,j-1} = U_{x,j-2}$ implies that $x \geq j - 1 \geq 2$, U is admissible.

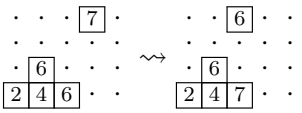
For the last two types of edges assume V has an outer column box and $V_{mj} < V_{j-1,j-1}$. Since V is weakly admissible, there exists a maximum row $x < j - 1$ with $V_{x,j-1} \leq V_{mj}$, and it must hold that $V_{mj} < V_{xj}$ and $V_{mj} < V_{x+1,j-1}$.

iC3b: Suppose $V_{x,j-1} = V_{mj}$, so $x \geq 2$, since V is admissible. There is an edge $V \rightsquigarrow U$ where U is formed from V by moving box (m, j) to $(m, j-1)$ and changing its value to be whichever of $V_{x-1,j-1}$ or $V_{x,j-2}$ is defined and larger.



We see that either $U_{x,j-2} = V_{x,j-2} \leq U_{m,j-1} < U_{x,j-1} = V_{x,j-1}$, if $U_{x,j-1}$ exists, or $U_{m,j-1} = U_{j,j-1}$, so U is weakly admissible. The edge $U \longrightarrow V$ is a column transition of type C3, so $\mathrm{col}(U) \overset{\mathrm{K}}{\approx} \mathrm{col}(V)$, and so $\mathrm{col}(U)$ is a symplectic Hecke word. If $U_{m,j-1} = U_{j-2,j-1}$ then $x - 1 = j - 2$, so $x = j - 1$. But x has to be strictly smaller than $j - 1$. If $U_{m,j-1} = U_{y,j-2}$, then $y = x \geq 2$. So U is admissible.

iC4: If $V_{x,j-1} < V_{mj}$, then there is an edge $V \rightsquigarrow U$ where U is formed from V by moving box $(x, j-1)$ to $(m, j-1)$ and then box (m, j) to $(x, j-1)$.

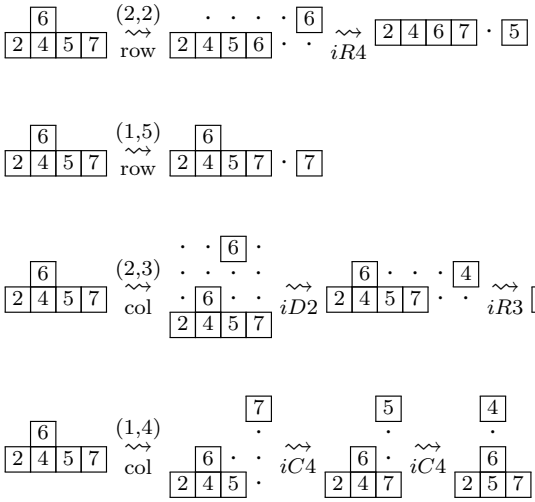


Since $U_{x,j-2} = V_{x,j-2} < V_{x,j-1} = U_{m,j-1} < V_{mj} = U_{x,j-1}$, it holds that U is weakly admissible. The edge $U \longrightarrow V$ is a column transition of type C4. Lemma 3.6 tells us, that $\mathrm{col}(U)$ is a symplectic Hecke word, as $\mathrm{row}(V)$ is one. If in this case $U_{m,j-1} = U_{j-2,j-1}$, then either $V_{x,j-1} = V_{mj}$ if $x = j - 2$ or $U_{j-2,j-1} = V_{j-2,j-1}$ if $x < j - 2$. The first case contradicts $V_{x,j-1} < V_{mj}$ directly, the second case imples

$V_{x,j-1} = U_{m,j-1} = V_{j-2,j-1}$ and contradicts $x < j-2$. If $U_{m,j-1} = U_{y,j-2}$, then $U_{m,j-1} = V_{y,j-2}$, but since $U_{m,j-1} = V_{x,j-1} < V_{x,j-2}$, y has to be strictly greater than x and so finally $y \geq 2$ and U is admissible.

This completes the definition of the edges for the inverse transition graph.

Example 25. We again take $P_{Sp}(265425)$ from Example 18 and look for some possible paths in the inverse transition graph:



This seems all but unique. We see how we find the correct path with the next definitions in Example 27.

Theorem 3.16. Let U and V be admissible insertion states. Then $U \longrightarrow V$ is a forward transition if and only if $V \rightsquigarrow U$ is an inverse transition. If V is terminal, then $U \xrightarrow{(i,j)} V$ is a row or a column transition if and only if $V \overset{(i,j)}{\underset{\text{row}}{\rightsquigarrow}} U$ or $V \overset{(i,j)}{\underset{\text{col}}{\rightsquigarrow}} U$ is an inverse transition.

The next thing we want to achieve is to get a second tableau for the tuple in Theorem 3.1. We want to define it in a way that helps us to find the right path in the inverse transition graph. For the main part of these paths, we need no help, since there is exactly one outgoing edge for every admissible non-terminal non-initial insertion state. Since the initial insertion states are the last tableaux on the paths in the inverse transition graph, we only need the second tableau to consider which is the correct first inverse transition, which edge start on the terminal insertion state. So we define now $Q_{Sp}(w)$ to conclude the type of the last edge of the forward transition graph:

Definition 3.17. For a symplectic Hecke word $w = w_1 w_2 \dots w_n$, we inductively define a set-valued tableau $Q_{Sp}(w)$. Let $Q_{sp}(\emptyset) = \emptyset$ and assume $n > 0$. Let (i, j) be the label of the last transition in the insertion path of $P_{Sp}(w_1 \dots w_{n-1}) \overset{\text{Sp}}{\leftarrow} w_n$. Form $Q_{Sp}(w)$ from $Q_{Sp}(w_1 \dots w_{n-1})$ as follows:

1. If the last transition is of type R1, then add n to box (i, j).
2. If the last transition is of type C1, then add n' to box (i, j).
3. If the last transition is of type R2, then add n to the last box in column $j - 1$.
4. If the last transition is of type C2, then add n' to the last box in row $i - 1$.

We call $Q_{Sp}(w)$ the **recording tableau** of w under symplectic Hecke insertion.

Lemma 3.15 ensures, that in every step $i-1 > 0$ and $j-1 > 0$, and that the $(i-1)$-th row and the $(j-1)$-th column exist in the already defined part of $Q_{Sp}(w)$, so $Q_{Sp}(w)$ is well-defined for any symplectic Hecke word w. By construction, $Q_{Sp}(w)$ is a standard shifted set-valued tableau of length $|Q_{Sp}(w)| = \mathrm{len}(w)$.

Example 26. We recall our first example of an insertion tableau we saw in Example 18. All the insertions ended with edges of type R1 or C1, except the last. So

$$Q_{Sp}(265425) = \begin{array}{|c|c|c|c|}\hline & 36 & & \\ \hline 1 & 2 & 4' & 5' \\ \hline \end{array}$$

In the same example we added $6, 2$ and 5, which delivers

$$Q_{Sp}(265425625) = \begin{array}{|c|c|c|c|}\hline & 36 & 79' & \\ \hline 1 & 2' & 4' & 5'8' \\ \hline \end{array}.$$

Definition 3.18. Let $z \in \mathcal{F}_\infty$ be an FPF-involution, P be an increasing shifted tableau, w be a word such that $\mathrm{row}(P)w \in \mathcal{H}_{\mathrm{Sp}}(z)$ is an FPF-involution word, Q be a standard set-valued tableau with the same shape as P and length $n > 0$. Q contains exactly one of n or n' and this number must appear in an inner corner (i, j), since Q is standard. Define V_1 to be the unique admissible shifted insertion state, such that

1. if $\{n\} = Q_{ij}$, then $P \overset{(i,j)}{\underset{\text{row}}{\rightsquigarrow}} V_1$ is an inverse transition.
2. if $\{n'\} = Q_{ij}$, then $P \overset{(i,j)}{\underset{\text{col}}{\rightsquigarrow}} V_1$ is an inverse transition.
3. if $\{n\} \subsetneq Q_{ij}$, then $P \overset{(r,j+1)}{\underset{\text{row}}{\rightsquigarrow}} V_1$ is an inverse transition, where r is the row of the unique outer corner of Q in column $j + 1$.
4. if $\{n'\} \subsetneq Q_{ij}$, then $P \overset{(i+1,s)}{\underset{\text{col}}{\rightsquigarrow}} V_1$ is an inverse transition, where s is the column of the unique outer corner of Q in row $i + 1$.

Now let $V_1 \rightsquigarrow V_2 \rightsquigarrow \cdots \rightsquigarrow V_l$ be the maximal directed path in the inverse transition graph. The last state V_l is initial, so it has the form $\hat{P} \oplus a$ for a shifted tableau $\hat{P}$ and a non-negative number $a \in \mathbb{N}$. Set $\hat{w} := aw$, and form $\hat{Q}$ by removing whichever of n or n' appears. With this we define

$$\mathrm{uninsert}(P, Q, w) := (\hat{P}, \hat{Q}, \hat{w}).$$

The set-valued tableau $\hat{Q}$ is standard with length $n-1$ and has the same shape as $\hat{P}$. By Theorem 3.16, we know that $P \rightsquigarrow V_1 \rightsquigarrow V_2 \rightsquigarrow \cdots \rightsquigarrow V_l = \hat{P} \oplus a$ is equivalent to $\hat{P} \oplus a \longrightarrow V_l \longrightarrow V_{l-1} \longrightarrow \cdots \longrightarrow V_1 = P$, so $P = \hat{P} \overset{\mathrm{Sp}}{\leftarrow} a$. By Proposition 3.13, it holds that $\mathrm{row}(P) \overset{\mathrm{Sp}}{\approx} \mathrm{row}(\hat{P})a$. Then $\mathrm{row}(\hat{P})\hat{w} = \mathrm{row}(\hat{P})aw \overset{\mathrm{Sp}}{\approx} \mathrm{row}(P)w \in \mathcal{H}_{\mathrm{Sp}}(z)$. Altogether, $(\hat{P}, \hat{Q}, \hat{w})$ has the same properties as (P, Q, w), so we can iterate uninsert:

Definition 3.19. Given all the requirements of the definition before, we define $w_{Sp}(P,Q)$ to be the word, such that

$$\mathrm{uninsert} \circ \cdots \circ \mathrm{uninsert}(P, Q, \emptyset) = (\emptyset, \emptyset, w_{Sp}(P,Q)),$$

with $|Q|$-times concatenated uninsert.

Example 27. We want to inverse the insertion of the word in Example 18. So we calculate $\mathit{uninsert} \circ \cdots \circ \mathrm{uninsert}(P_{Sp}(265425), Q_{Sp}(265425), \emptyset)$:

The highest number in the recording tableau (seen in Example 26) is 6 in the box at position $(1,4)$, so the first inverse edge has to be of type iC2 at the outer corner of the second row:

$$P_{Sp}(265425) = \begin{array}{|c|c|c|c|} \hline & & & \boxed{6} \\ \hline 2 & 4 & 5 & 7 \\ \hline \end{array} \overset{(2,4)}{\underset{\mathrm{col}}{\rightsquigarrow}} \begin{array}{|c|c|c|c|} \hline & & & \boxed{6} \\ \hline 2 & 4 & 5 & 7 \\ \hline \end{array} \cdot \boxed{6} \rightsquigarrow \begin{array}{|c|c|c|c|} \hline & & & \boxed{6} \\ \hline 2 & 4 & 5 & 7 \\ \hline \end{array} \cdot \boxed{5} = P_{Sp}(26542) \oplus 5,$$

so

$$\mathrm{uninsert}\left(\begin{array}{|c|c|c|c|} \hline & & & \boxed{6} \\ \hline 2 & 4 & 5 & 7 \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline & & & \boxed{36} \\ \hline 1 & 2 & 4' & 5' \\ \hline \end{array}, \emptyset \right) = \left(\begin{array}{|c|c|c|c|} \hline & & & \boxed{6} \\ \hline 2 & 4 & 5 & 7 \\ \hline \end{array}, \begin{array}{|c|c|c|c|} \hline & & & \boxed{3} \\ \hline 1 & 2 & 4' & 5' \\ \hline \end{array}, 5 \right)$$

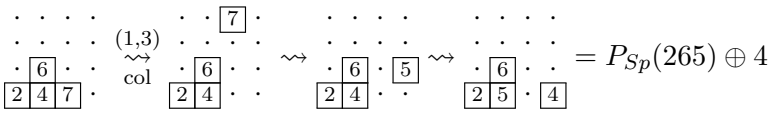
which is $(P_{Sp}(26542), Q_{Sp}(26542), 5)$.

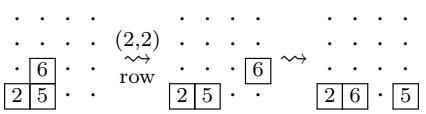
For the first step for the next uninsertion, we have to look onto $Q_{Sp}(26542)$. The highest number is $5'$ in box $(1,4)$, so the first edge for $\mathrm{uninsert}(P_{Sp}(26542), Q_{Sp}(265425), 3)$ has to be of type iC1 at position $(1,4)$:

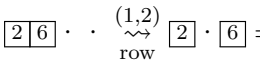
$$P_{Sp}(26542) = \begin{array}{|c|c|c|c|} \hline \cdot & \cdot & \cdot & \boxed{7} \\ \hline \cdot & \boxed{6} & \cdot & \cdot \\ \hline 2 & 4 & 6 & 7 \\ \hline \end{array} \overset{(1,4)}{\underset{\mathrm{col}}{\rightsquigarrow}} \begin{array}{|c|c|c|c|} \hline \cdot & \cdot & \cdot & \boxed{7} \\ \hline \cdot & \boxed{6} & \cdot & \cdot \\ \hline 2 & 4 & 6 & \cdot \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|c|c|} \hline \cdot & \cdot & \boxed{6} & \cdot \\ \hline \cdot & \boxed{6} & \cdot & \cdot \\ \hline 2 & 4 & 7 & \cdot \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|c|c|} \hline \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \boxed{6} & \cdot & \boxed{4} \\ \hline 2 & 4 & 7 & \cdot \\ \hline \end{array} \rightsquigarrow \begin{array}{|c|c|c|c|} \hline \cdot & \cdot & \cdot & \cdot \\ \hline \cdot & \boxed{6} & \cdot & \cdot \\ \hline 2 & 4 & 7 & \cdot \\ \hline \end{array} \cdot \boxed{2} = P_{Sp}(2654) \oplus 2.$$

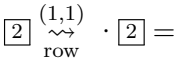
So, $\mathrm{uninsert}(P_{Sp}(26542), Q_{Sp}(26542), 5) = \mathrm{uninsert}(P_{Sp}(2654), Q_{Sp}(2654), 25)$.

And further we get:









So finally, we see that we can inverse the insertion from Example 18:

$$\text{uninsert} \circ \cdots \circ \text{uninsert}\left(\begin{array}{|c|c|c|c|}\hline & \multicolumn{1}{c}{6} & & \\ \hline 2 & 4 & 5 & 7 \\ \hline \end{array}, \begin{array}{|c|c|c|c|}\hline & \multicolumn{1}{c}{36} & & \\ \hline 1 & 2 & 4' & 5' \\ \hline \end{array}, \emptyset\right) = (\emptyset, \emptyset, 265425)$$

and so we calculated $w_{Sp}(PSp(265425), QSp(265425), \emptyset) = 265425$.

We want to define the inversion of uninsert:

Definition 3.20. As in the definition for uninsert, let P be an increasing shifted tableau, Q be a standard set-valued tableau with the same shape as P and $|Q| = n - 1 \geq 0$, and let $w = w_1 w_2 \dots w_m$ be a word with $m > 0$ and $row(P)w \in \mathcal{H}_{\mathrm{Sp}}(z)$. Let $P = P_{Sp}(v)$ be an insertion tableau for a symplectic Hecke word v under symplectic Hecke insertion, $Q = Q_{Sp}(v)$ be his recording tableau. Let $\check{P} = P_{Sp}(vw_1)$ be the insertion tableau, $\check{Q} = Q_{Sp}(vw_1)$ the recording tableau of vw_1 under symplectic Hecke insertion, and $\check{w} = w_2 \dots w_m$. We define

$$\text{insert}(P, Q, w) := (\check{P}, \check{Q}, \check{w}).$$

Since $\mathrm{row}(\check{P})\check{w} \overset{\mathrm{Sp}}{\approx} \mathrm{row}(P)w_1 w$ by Theorem 3.8 and $\mathrm{row}(P)w_1 w = \mathrm{row}(P)w \in \mathcal{H}_{\mathrm{Sp}}(z)$, it holds that $\mathrm{row}(\check{P})\check{w} \in \mathcal{H}_{\mathrm{Sp}}(z)$. So again, we can iterate the operation and get for $w \in \mathcal{H}_{\mathrm{Sp}}(z)$, that

$$\text{insert} \circ \cdots \circ \text{insert}(\emptyset, \emptyset, w) = (P_{Sp}(w), Q_{Sp}(w), \emptyset),$$

with $\mathrm{len}(w)$-times concatenated insert.

We have everything to proof Theorem 3.1 now.

Proof. Let T_n^m be the set of triples (P, Q, w), where P is an increasing shifted tableau, Q is a standard set-valued tableau of length n with the same shape as P and w is a word of length m, such that $\mathrm{row}(P)w \in \mathcal{H}_{\mathrm{Sp}}(z)$. We can see, that insert : $T_n^{m+1} \to T_{n+1}^m$ and uninsert : $T_{n+1}^m \to T_n^{m+1}$.

The work to show that they are inverses is already done: By their definition, uninsert only combines edges in the inverse transition graph and insert combines edges of the forward transition graph. We already saw that every insertion state on the path of insert is admissible. Since the inverse transition graph is defined on admissible insertion states, every insertion state on the path of uninsert has to be admissible too. So we can use Theorem 3.8, which shows that the two paths in $\mathrm{uninsert}(P, Q, w)$ and $\mathrm{insert}(\mathrm{uninsert}(P, Q, w))$, and in $\mathrm{insert}(P, Q, w)$ and $\mathrm{uninsert}(\mathrm{insert}(P, Q, w))$ are reverse. That implies that uninsert and insert are inverse, so the symplectic Hecke insertion is the bijection we are looking for.

Additionally, suppose $w \in \mathcal{R}_{\mathrm{FPF}}(z)$ is an FPF-involution word. Since we only add a maximum of one box to P for every letter in w, we must have $\mathrm{len}(w) \geq |P_{Sp}(w)|$. By Lemma 3.9, we know that $w \overset{\mathrm{Sp}}{\approx} \mathrm{row}(P_{Sp}(w))$, so $\mathrm{row}(P_{Sp}(w) \in \mathcal{H}_{\mathrm{Sp}}(z)$. But the FPF-involution words are the symplectic Hecke words of minimal length, so $\mathrm{len}(w) = |P_{Sp}(w)|$. It follows, that Q has to be marked, since in the last step of an insertion we can never omit a box with (R2) or (C2).

To proof this property for the inverse, assume $\mathrm{row}(P) \in \mathcal{R}_{\mathrm{FPF}}$ and Q is a marked tableau of the same shape. Since every box of Q has exactly one entry, we only use (iC1) or (iR1) in the first step for uninsert, so for every box of P, we add one more letter to $w_{Sp}(P, Q)$, so $|w_{Sp}(P, Q)| = |P|$ and $w_{Sp}(P, Q) \in \mathcal{R}_{\mathrm{FPF}}(z)$. □

3.3 Semistandard variant

Before we can use this bijection as intended, we have to refine it in this section and look onto another bijection, the semistandard symplectic Hecke insertion. We use this insertion for the K-theoretic P-functions in the next chapter. This primarily follows section 4.1 of [5].

First we analyse the bumping path from Definition 3.3. We already saw that the types of transitions can only appear in a fixed order: there is a maximum of one diagonal transition, before it there can only be row transitions, after it there can only be column transitions.

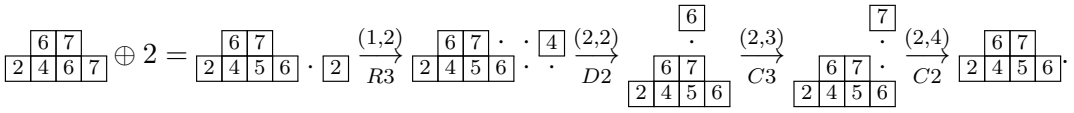
Definition 3.21. Let $(i_1, j_1), (i_2, j_2), \ldots, (i_l, j_l)$ be a bumping path, resulting from inserting a non-negative integer $a \in \mathbb{N}$ into an increasing shifted tableau T. We refer to the positions up to and including the first diagonal position as **row-bumped positions** and to any subsequent position as **column-bumped position**.

Remark 3.22. We have to be a bit careful with the intuition of this definition: The diagonal position is either obtained by D2, D3, or D4, or it is obtained by C2, which has to appear directly after D1 appeared. Since D1 creates $(i, i+1)$ in the bumping-path, the last row-bumped position can be a column transition.

For an example we can look onto the bumping path from inserting 5 into $P_{Sp}(26542565)$, which we saw in Example 18. In spite of the position from the diagonal transition at the second edge, the third position in the bumping path is the last row-bumped position (created from a column transition).

Another fact we can see in the definitions of the edges of the forward transition graph is that all the row transitions create a bumping path position in the row of the outer box and all the column transitions create a bumping path position in the column of the outer corner. So we know that i_t has to be equal to t, if (i_t, j_t) is a row-bumped position, and j_t has to be t if it is a column-bumped position.

Example 28. We look onto the bumping-path from the insertion of 2 into $PSp(2654256)$, as we have seen it in Example 18.



So the bumping-path in this case is $(1,2),(2,2),(2,3),(2,4)$, while the first two positions are row-bumped and the last two positions are column-bumped. We see that $(i_1, i_2, j_3, j_4) = (1,2,3,4)$ and the last row-bumped position, which has to be on the diagonal, has $i_2 = j_2 = 2$.

Proposition 3.23. Suppose T is an increasing shifted tableau and $a, b \in \mathbb{N}$ are integers with $a \leq b$, such that $\mathrm{row}(T)ab \in \mathcal{H}_{\mathrm{Sp}}(z)$ is a symplectic Hecke word. We refer to $U := T \overset{\mathrm{Sp}}{\leftarrow} a$ as the first bumping path and to $V = U \overset{\mathrm{Sp}}{\leftarrow} b$ as the second one. Then:

1. Suppose the i-th element of the first path is row-bumped and the second path has length at least i. Then the i-th elements of both paths are row-bumped and in row i, and the i-th element of the first path is weakly left of the i-th element of the second path.

2. If the last position in the first path is row-bumped and occurs in column j, then the last position in the second path is row-bumped and occurs in column k, where $j \leq k$.

3. Suppose the i-th element of the second path is column-bumped. Then the first path has length at least i, the i-th elements of both paths are column-bumped and in column i, and the i-th element of the first path is weakly below of the i-th element of the second path.

4. If the last position in the second path is column-bumped and occurs in row j, then the last position in the first path is column-bumped and occurs in row i, where $i \leq j$, and in a weakly lefter column.

Definition 3.24. The **descent set** of a word $w = w_1 w_2 \dots w_n$ is

$$\mathrm{Des}(w) = \{i \in [n-1] : w_i > w_{i+1}\}.$$

The descent set of a standard shifted set-valued tableau T with length $|T| = n$ is

$$\mathrm{Des}(T) := \{i \in [n-1] : \begin{cases} i \text{ and } (i+1)' \text{ both appears in } T, \text{ or} \\ i \text{ appears in } T \text{ and } i+1 \text{ appears in } T \text{ in a row above of } i, \text{ or} \\ i' \text{ appears in } T \text{ and } (i+1)' \text{ appears in } T \text{ in a column right of } i'. \end{cases}$$

Example 29. We take the word from Example 18 again:

$$\mathrm{Des}(265425) = \{2, 3, 4\}.$$

Now we look again onto $Q_{Sp}(265425)$, which we already calculated in Example 26:

$$Q_{Sp}(265425) = \begin{array}{|c|c|c|c|} \hline & \multicolumn{1}{c|}{36} & & \\ \hline 1 & 2 & 4' & 5' \\ \hline \end{array}.$$

We can check that $\mathrm{Des}(Q_{Sp}(4232143)) = \{2, 3, 4\}$, since: 2 and 3 appear and 3 is in a higher row, 3 and $4'$ are in the tableau, and $4'$ and $5'$ appear and $5'$ is in a column right of $4'$.

Theorem 3.25. If w is a symplectic Hecke word then $\mathrm{Des}(w) = \mathrm{Des}(Q_{Sp}(w))$.

Remark 3.26. Let T be a standard shifted set-valued tableau. If we create a new tableau S from T by putting all unprimed entries of T in the same box in S and moving every primed entry in box $T_{x,y}$ to $S_{y,x}$, then i is a descent if and only if the row of S which is containing i is strictly below the row of the row in S which is containing $i+1$.

Definition 3.27. For a symplectic Hecke word $w \in \mathcal{H}_{\mathrm{Sp}}(z)$ of length m and for a weakly increasing factorisation $i = (i_1 \leq i_2 \leq \cdots \leq i_m)$ of w we define $Q_{Sp}(w, i)$ to be the shifted weak set-valued tableau formed from $Q_{Sp}(w)$ by replacing j by i_j and j' by i'_j for each $j \in [m]$.

Example 30. We choose the weakly increasing factorisation $i = (112344)$ for the word $w = 265425$. Then

$$Q_{Sp}(w, i) = \begin{array}{|c|c|c|c|} \hline & \multicolumn{1}{c|}{24} & & \\ \hline 1 & 1 & 3' & 4' \\ \hline \end{array}.$$

Definition 3.28. Let Q be a semistandard shifted weak set-valued tableau. We define the **standardisation** $\mathrm{st}(Q)$ of Q by changing the entries of Q in the following way:

- First replacing all 1s in Q from left to right by $1, 2, \ldots i$.
- Then replacing all $1'$s in Q from bottom to top by $(i+1)', (i+2)', \ldots, j'$.
- Then replacing all 2s in Q from left to right by $j+1, j+2, \ldots, k$.
- Then replacing all $2'$ in Q from bottom to top by $(k+1)', (k+2)', \ldots, l'$.

- And so on, that means replacing the primed and unprimed numbers in increasing order by counting either from left to right if the number is unprimed, or from bottom to top if the number is primed, and prime them if the former number was primed too. Continue this process till all entries in Q are replaced.

Example 31. We calculate the standardisation

$$\mathrm{st}(Q_{Sp}(265425, 112344)) = \mathrm{st}\left(\begin{array}{|c|c|c|c|}\hline & \boxed{24} & & \\ \hline 1 & 1 & 3' & 4' \\ \hline \end{array}\right) = \begin{array}{|c|c|c|c|}\hline & \boxed{36} & & \\ \hline 1 & 2 & 4' & 5' \\ \hline \end{array}.$$

And we see that we obtain $Q_{Sp}(265425)$.

Theorem 3.29. Let $z \in \mathcal{F}_\infty$. The map $(w, i) \mapsto (P_{Sp}(w), Q_{Sp}(w, i))$ is a bijection from weakly increasing factorisations of symplectic Hecke word to pairs (P, Q), where P is an increasing shifted tableau with $\mathrm{row}(P) \in \mathcal{H}_{\mathrm{Sp}}(z)$ and Q is a semistandard shifted weak set-valued tableau with the same shape as P. Moreover, the bijection is weight-preserving in the following sense: $wt_{Q_{Sp}(w,i)} = wt_i$.

Proof. Let (w, i) be a weakly increasing factorisation of a symplectic Hecke word. The construction of $Q_{Sp}(w, i)$ implies that there are as many j or j' in $Q_{Sp}(w, i)$ as there are j in the weakly increasing factorisation i. So the map is weight-perserving.

If we take a streak of the same number in i, the indices are all not descents. Theorem 3.25 shows that we must have a sequence of the same length of increasing numbers, which are elements of $Q_{Sp}(w)$ and which are not descents too. Let us take the maximal streak of an number $h \in \mathbb{N}$, so $i_t = h$ if and only if $t \in [j : j + b]$ for some $j, b \in \mathbb{N}$. Since the corresponding elements in $Q_{Sp}(w)$ are no descents, it holds that we can divide the numbers in an earlier sequence of primed numbers and a latter sequence of unprimed ones, that means there exists an $a \leq b$ with $j', (j+1)', \ldots (j+a)', j+a+1, \ldots, j+b \in Q_{Sp}(w)$. Moreover, two primed entries of the sequence cannot be in different boxes in the same row, while two unprimed entries cannot be in different boxes in the same column of $Q_{Sp}(w)$. So if we change the entries to form $Q_{Sp}(w, i)$, we cannot get any unprimed number in more than one box of any column and we cannot get any primed number in more than one box of any row. Since $Q_{Sp}(w)$ is standard, increasing and does not have any primed entries on the diagonal, it follows that $Q_{Sp}(w, i)$ has to be weakly increasing and $Q_{Sp}(w, i)$ does not have any primed entries on the diagonal too. So $Q_{Sp}(w, i)$ is semistandard and the map is well-defined.

To define the inverse we have to reconstruct $Q_{Sp}(w)$ and i from an given $Q_{Sp}(w, i)$. But $Q_{Sp}(w)$ is the standardisation of $Q_{Sp}(w, i)$. To get the correct factorisation i, we define $i^Q = (i_1^Q \leq i_2^Q \leq \cdots \leq i_m^Q)$ with $m = |Q|$, by $i_j^Q := a$ if a or a' appears in Q and changes to j or j' in $\mathrm{st}(Q)$.

Let (w, i) be a weakly increasing factorisation of a symplectic Hecke word. By Theorem 3.25 we can see that every semistandard shifted weak set-valued tableau, whose standardisation is $Q_{Sp}(w)$, arises as $Q_{Sp}(w, i)$ for the choice of some factorisation i. It follows from Theorem 3.1, that the map is surjective. Since we can recover $Q_{Sp}(w)$ and i from $Q_{Sp}(w, i)$ as descried, we can also recover w by Theorem 3.1, so the given map is injective. □

4 Polynomials

In this chapter, we use the semistandard symplectic Hecke insertion (especially Theorem 3.29) to prove that the K-theoretic Schur P-functions have positive structure constants, as stated in the following theorem. This involves working out the bijective proof outlined in section 1.2 of [4]. Afterwards, we see a bigger example.

For a given strict partition λ we define the K-theoretic Schur P-function $\mathrm{GP}_\lambda = \sum_T x^T$ as the generating function of all semistandard set-valued shifted tableaux of shape λ. We want to see now how the symplectic Hecke insertion helps us as argument to prove the following theorem.

Theorem 4.1. Let λ, μ be strict partitions. It holds that

$$\mathrm{GP}_\lambda \cdot \mathrm{GP}_\mu = \sum_\nu e^\nu_{\lambda\mu} \, \mathrm{GP}_\nu,$$

while $e^\nu_{\lambda\mu}$ is the number of insertion tableaux with shape ν and row reading word in $\mathcal{H}_{\mathrm{Sp}}(z_\lambda \times z_\mu)$.

Furthermore, we defined $\mathrm{KP}_\lambda = \sum_T x^T$ as the generating function of all weak set-valued shifted tableaux of shape λ with no primes on the diagonal at the end of section 2.3. These functions help us, since it is already known, that the latter generating function is related to the Schur P-functions by the automorphism of the algebra of symmetric functions ω which maps the Schur functions $s_\lambda \mapsto s_{\lambda^T}$. It holds that $\mathrm{GP}_\lambda = \omega(\mathrm{KP}_\lambda)$. We refer to [6, Corollary 6.6] for details.

So to prove Theorem 4.1, we can prove instead the following lemma.

Lemma 4.2. Let λ, μ be two strict partitions. It holds that

$$\mathrm{KP}_\lambda \cdot \mathrm{KP}_\mu = \sum_\nu e^\nu_{\lambda\mu} \, \mathrm{KP}_\nu$$

for some numbers $e^\nu_{\lambda\mu} \in \mathbb{N}^0$.

If this is proven, we know that

$$\mathrm{GP}_\lambda = \omega(\mathrm{KP}_\lambda) = \omega(\sum_\nu e^\nu_{\lambda\mu} \, \mathrm{KP}_\nu) = \sum_\nu e^\nu_{\lambda\mu} \omega(\mathrm{KP}_\nu) = \sum_\nu e^\nu_{\lambda\mu} \, \mathrm{GP}_\nu \, .$$

Definition 4.3. Let $z \in \mathcal{F}_\infty$ be an FPF-involution. We define

$$\mathrm{KP}_z := \sum_{(w,i) \in \mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(z))} x^{(w,i)},$$

where $x^{(w,i)} = \prod_j x_j^{\mathrm{wt}_i(j)} = \prod_i x_j^{\mathrm{len}(w^j)} = \text{for } (w^1, w^2, \dots) \mapsto (w, i)$.

Example 32. We want to calculate some factors of $\mathrm{KP}_{s_1s_2\theta s_2s_1}$. With the rules of $\overset{\mathrm{Sp}}{=}$ we can compute the set of symplectic Hecke words of $s_1s_2\theta s_2s_1$ with length lower than 4:

$$\{23, 21, 223, 233, 221, 211, 231, 213\}.$$

To calculate the factor of a monomial $\prod x_i^{a_i}$, we can just count the number of pairs $(w, j) \in \mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(s_1s_2\theta s_2s_1))$, such that i appears in the weakly increasing sequence j exactly a_i times.

$(23,11)$	$(21,12)$	$(23,22)$	$(223,111)$	$(221,112)$	$(211,122)$	$(223,222)$
	$(23,12)$		$(233,111)$	$(223,112)$	$(223,122)$	$(233,222)$
				$(233,112)$	$(233,122)$	
				$(231,112)$	$(213,122)$	

This are all possible weakly increasing factorisations of words in $\mathcal{H}_{\mathrm{Sp}}(s_1s_2\theta s_2s_1)$, where the factorisation uses only 1 and 2 and the lenth of the word is lower than 4. So we get

$$\mathrm{KP}_{s_1s_2\theta s_2s_1} = 1 \cdot x_1^2 + 2 \cdot x_1x_2 + 1 \cdot x_2^2 + 2 \cdot x_1^3 + 4 \cdot x_1^2x_2 + 4 \cdot x_1x_2^2 + 2 \cdot x_2^3 + \dots$$

Lemma 4.4. For a strict partition λ it holds that

$$\mathrm{KP}_z = \sum_{S \in \{P_{Sp}(w) : w \in \mathcal{H}_{\mathrm{Sp}}(z)\}} \mathrm{KP}_{\mathrm{shape}(S)}\,.$$

Proof. Since the semistandard variant of the symplectic Hecke insertion is weight preserving, we can see that $x^{(w,i)} = \prod_j x_j^{wt_i(j)} = x^{Q_{Sp}(w,i)}$. So $\mathrm{KP}_z = \sum_{(w,i)\in\mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(z))} x^{Q_{Sp}(w,i)}$. Since we know that the tuples (w, i) of this form are in bijection with $(P_{Sp}(w), Q_{Sp}(w, i))$, we can rewrite this sum as $\mathrm{KP}_z = \sum_{S\in\{P_{Sp}(w):w\in\mathcal{H}_{\mathrm{Sp}}(z)\}} \sum_{\mathcal{T}} x^T$, where the inner generating function is over the set $\mathcal{T}$ of all semistandard weak set-valued tableaux with the same shape as S. So $\mathrm{KP}_z = \sum_{S\in\{P_{Sp}(w):w\in\mathcal{H}_{\mathrm{Sp}}(z)\}} \mathrm{KP}_{\mathrm{shape}(S)}$. □

Lemma 4.5. Let $y, z \in \mathcal{F}_\infty$. It holds that

$$\mathrm{KP}_y \cdot \mathrm{KP}_z = \mathrm{KP}_{y\times z}\,.$$

Proof. We saw a bijection $\mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(y)) \times \mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(z)) \longrightarrow \mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(y \times z))$ at the end of section 2.1. With that we are able to see, that

$$\begin{aligned}
\mathrm{KP}_y \cdot \mathrm{KP}_z &= \sum_{(v,i)\in\mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(y))} \prod_i x_i^{\mathrm{len}(v_i)} \sum_{(w,j)\in\mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(z))} \prod_j x_j^{\mathrm{len}(w_j)} \\
&= \sum_{\mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(y))\times\mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(z))} \prod_{i,j} x_i^{\mathrm{len}(v_i)} x_j^{\mathrm{len}(w_j)} \\
&= \sum_{\mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(y))\times\mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(z))} \prod_{i} x_i^{\mathrm{len}(v_i)+\mathrm{len}(w_j)} \\
&= \sum_{\mathrm{Incr}(\mathcal{H}_{\mathrm{Sp}}(y)\times\mathcal{H}_{\mathrm{Sp}}(z))} \prod x_i^{\mathrm{len}(v_i\check{w}_i)} \\
&= \mathrm{KP}_{y\times z}\,.
\end{aligned}$$

□

Remark 4.6. The last fact we need is that for every λ which has a maximum of $n-1$ columns, there is a formular for an FPF-involution $z_\lambda \in \mathcal{F}_n$ such that $\mathrm{KP}_\lambda = \mathrm{KP}_{z_\lambda}$ [3].

Since we do not want to proof it here, we want to get convinced by an example:

To calculate a factor of the monomial $\prod x_i^{a_i}$ in $\mathrm{KP}_{(1)}$, we have count the number of semistandard weak set-valued shifted tableaux of shape (1) in which i or i' appears a_i times. But to get a semistandard tableau, we can fill the single box of the young diagram exactly one time with the multiset which contains a_i times i, since we are not allowed to prime anything in the diagonal box. So the factor of every monomial in $\mathrm{KP}_{(1)}$ is 1.

To get this polynomial by calculating KP_z for a fix-point-free involution $z \in \mathcal{F}_\infty$, we need a set of symplectic Hecke words, in which every weakly increasing sequence of numbers appear exactly once as weakly increasing factorisation for a word in the set. This is realised if we take the symplectic Hecke words of $\mathcal{H}_{\mathrm{Sp}}(s_2\theta s_2) = \{2, 22, 222, 2222, 22222\dots\}$. Since these words never decrease, we can take any weakly increasing sequence as weakly increasing factorisation. And since there is only one word of a specific length, we can choose every of these sequences exactly once. So $\mathrm{KP}_{(1)} = \mathrm{KP}_{s_2\theta s_2}$.

Now we want to calculate some factors of $\mathrm{KP}_{(2)}$:

$$\mathrm{KP}_{(2)} = \quad 1x_1^2 \quad +2x_1x_2 \quad +1x^2 \quad +2x_1^3 \quad +4x_1^2x_2 \quad +4x_1x_2^2 \quad +\dots$$

1	1		1	2		2	2		11	1		11	2		12	2			
			1	2′					1	11		11	2′		1	22			
												1	12		1	2′2			
												1	12′		1	2′2′			

We see that this $\mathrm{KP}_{(2)}$ has the same factors as the polynomial in Example 32. And indeed, $z_{(2)} = s_1s_2\theta s_2s_1$.

Proof for Lemma 4.2. We summarise all we have learned in this section, and get:

$$\mathrm{KP}_\lambda \cdot \mathrm{KP}_\mu \overset{4.6}{=} \mathrm{KP}_{z_\lambda} \cdot \mathrm{KP}_{z_\mu} \overset{4.5}{=} \mathrm{KP}_{z_\lambda \times z_\mu} \overset{4.4}{=} \sum_{T \in \{P_{Sp}(w) : w \in \mathcal{H}_{\mathrm{Sp}}(z_\lambda \times z_\mu)\}} \mathrm{KP}_{shape(T)} = \sum_\nu e^\nu_{\lambda\mu} \mathrm{KP}_\nu,$$

where $e^\nu_{\lambda\mu}$ is the number of insertion tableaux with shape ν and row reading word in $\mathcal{H}_{\mathrm{Sp}}(z_\lambda \times z_\mu)$, since the symplectic Hecke insertion is bijective. □

Example 33. We want to check whether the equation holds for $\mathrm{KP}_{(1)} \cdot \mathrm{KP}_{(2)}$.

We already calculated at least a few factors of these polynomials:

$$\begin{aligned}\mathrm{KP}_{(1)} &= 1x_1 + 1x_2 + 1x_1^2 + 1x_1x_2 + 1x_2^2 + 1x_1^3 + 1x_1^2x_2 + 1x_1x_2^2 + 1x_2^3 + \dots \\ \mathrm{KP}_{(2)} &= \qquad 1x_1^2 + 2x_1x_2 + 1x_2^2 + 2x_1^3 + 4x_1^2x_2 + 4x_1x_2^2 + 2x_2^3 + \dots\end{aligned}$$

So we can calculate

$$\mathrm{KP}_{(1)} \cdot \mathrm{KP}_{(2)} = x_1^3 + 3x_1^2x_2 + x_2^3 + 3x_1^4 + 9x_1^3x_2 + 12x_1^2x_2^2 + 9x_1x_2^3 + 3x_2^4 + \dots$$

Now we want to calculate $\sum_\nu e^\nu_{(1)(2)} \mathrm{KP}_\nu$. We already saw that $z_{(1)} = s_2\theta s_2$, while we claimed that $z_{(2)} = s_1s_2\theta s_2s_1$. With the bijection we defined at the end of section 2.1, we can see that $((2,1),(21,12)) \longmapsto (265,112)$, and thus we know that 265 has to be a symplectic Hecke word in $\mathcal{H}_{\mathrm{Sp}}(z_{(1)} \times z_{(2)})$. Since this set has to be an equivalence class of $\overset{\mathrm{Sp}}{\equiv}$, we can use its rules to find the other symplectic Hecke words: $\mathcal{H}_{\mathrm{Sp}}(z_{(1)} \times z_{(2)}) = \{265, 625, 652, 672, 627, 267, 6257 \dots\}$.

By calculating the insertion tableaux of this set, we get

$$\begin{array}{|c|c|}\hline \multicolumn{1}{|c|}{} & 6 \\ \hline 2 & 5 \\ \hline \end{array} = P(265) = P(625) = P(2265) = P(2625) = P(2665) = \dots,$$

$$\begin{array}{|c|c|c|}\hline 2 & 6 & 7 \\ \hline \end{array} = P(652) = P(672) = P(627) = P(267) = P(6652) = \dots,$$

$$\begin{array}{|c|c|c|}\hline \multicolumn{1}{|c|}{} & 6 & \multicolumn{1}{c|}{} \\ \hline 2 & 5 & 7 \\ \hline \end{array} = P(2652) = P(6253) = P(6525) = P(6725) = P(6275) = \dots.$$

Especially, no other insertion tableau appears, independently how long the words are. So $e^{(2,1)}_{(1)(2)} = e^{(3)}_{(1)(2)} = e^{(3,1)}_{(1)(2)} = 1$ and every other $e^\nu_{(1)(2)} = 0$, such that

$$\mathrm{KP}_{(1)} \cdot \mathrm{KP}_{(2)} = \mathrm{KP}_{(2,1)} + \mathrm{KP}_{(3)} + \mathrm{KP}_{(3,1)}\,.$$

To calculate them, we are again interested in semistandard weak set-valued shifted tableaux, as we calculated terms of $\mathrm{KP}_{(2,1)}$ in Example 16 already.

$$\mathrm{KP}_{(2,1)} = 1 \cdot x_1^2x_2 + 1 \cdot x_1x_2^2 + 2 \cdot x_1^3x_2 + 2 \cdot x_1x_2^3 + 3 \cdot x_1^2x_2^2 + \dots$$

The tableaux of shape (3), which are interesting for the monomials of degree smaller or equal to 4, are:

$$\begin{array}{|c|c|c|}\hline 1&1&1\\\hline\end{array}\ \text{and}\ \begin{array}{|c|c|c|}\hline 2&2&2\\\hline\end{array}\qquad \begin{array}{|c|c|c|}\hline 1&1&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&1&2'\\\hline\end{array}\qquad \begin{array}{|c|c|c|}\hline 1&2&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&2'&2\\\hline\end{array}$$

$$\begin{array}{|c|c|c|}\hline 11&1&1\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&11&1\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&1&11\\\hline\end{array}\ \text{and}\ \begin{array}{|c|c|c|}\hline 22&2&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 2&22&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 2&2&22\\\hline\end{array}$$

$$\begin{array}{|c|c|c|}\hline 11&1&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 11&1&2'\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&11&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&11&2'\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&1&12\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&1&12'\\\hline\end{array}$$

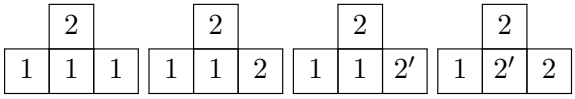
$$\begin{array}{|c|c|c|}\hline 11&2&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 11&2'&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&12&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&12'&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&1&22\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&1&2'2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&1&2'2'\\\hline\end{array}$$

$$\begin{array}{|c|c|c|}\hline 12&2&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&22&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&2'2&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&2'2'&2\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&2&22\\\hline\end{array}\ \begin{array}{|c|c|c|}\hline 1&2'&22\\\hline\end{array}$$

So we see that

$$\mathrm{KP}_{(3)} = x_1^3 + 2x_1^2x_2 + x_1x_2^2 + x_2^3 + 3x_1^4 + 6x_1^3x_2 + 7x_1^2x_2^2 + 6x_1x_2^3 + 3x_2^4 + \dots.$$

And finally, we have to look for semistandard weak set-valued shifted tableaux of shape $(3,1)$:



So $\mathrm{KP}_{(3,1)} = x_1^3x_2 + 2x_1^2x_2^2 + x_1x_2^3 + \dots$ and alltogether

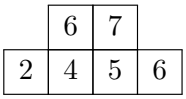
$$\begin{aligned}\mathrm{KP}_{(2,1)} + \mathrm{KP}_{(3)} + \mathrm{KP}_{(3,1)} &= x_1^3 + 3x_1^2x_2 + x_2^3 + 3x_1^4 + 9x_1^3x_2 + 12x_1^2x_2^2 + 9x_1x_2^3 + 3x_2^4 + \dots \\ &= \mathrm{KP}_{(1)} \cdot \mathrm{KP}_{(2)}\,.\end{aligned}$$

5 Algorithms in Sage

This chapter contains Sage code that implements the algorithms discussed earlier. The chapter is divided into several sections. The first section contains code related to words and relations, including a code that returns a list of words in a given $\overset{\mathrm{Sp}}{\equiv}$ equivalence class of a fixed length. The second section contains code related to tableaux, including a code that checks if a tableau is weakly admissible. The next two sections contain the code for (semistandard) symplectic Hecke insertion and its inverse. Afterward, there is a section that defines a way to return the visualization of a tableau and useful documentations. In the last section are examples of requests and their corresponding output.

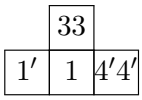
Before we see the code there are a few remarks to mention the differences between the notation in the paper and in the code.

- Words are realised by a list of its letters.
- Primed numbers are realized by subtracting 0.5. So $5'$ is represented by 4.5.
- Marked shifted tableaux (especially P_{Sp}) are realized by a two dimensional list, where the inner lists each represent one row of the tableaux. So for example



is saved as $[[2,4,5,6],[6,7]]$.

- (Weak) set-valued shifted tableaux are realized by three dimensional lists, where the second-level lists also represent the rows, where the third-level lists represents the (multi-)sets of each box. So for example



is saved as $[[[0.5],[1],[3.5,3.5]],[[3,3]]]$.

- Due to the representation of tableaux, there are two kinds of positions, we are using. **Absolute** positions are the numeration, how we did it in this paper too.

Relative positions are the places in the more dimensional lists. So for example

$$T = \begin{array}{|c|c|c|c|}\hline & 5 & 6 & \\ \hline 1 & 2 & 3 & 4 \\ \hline \end{array} \text{ saved as } [[1,2,3,4],[5,6]]$$

has 5 on the absolute position $(2,2)$ but in the relative position $(2,1)$, since it is in the first entry of the second list.

- Furthermore, the lists start to count with 0, so if we want to get the entry 5 in the previous tableau, we need to ask for $T[1][0]$.
- Outer boxes are represented by a list of length 3. The first entry is the value, the latter two are the absolute position in the tableau. So for example

$$\begin{array}{cccc} \cdot & \boxed{6} & \cdot & \boxed{5} \\ \boxed{2} & \boxed{5} & \cdot & \cdot \end{array}$$

is represented by the tableau $[[2,5],[6]]$ with outer box $[5,3,1]$

5.1 Relations and words

```
# adds word to list, if its not in there already
def add_for_list(list, word):
    if not word in list:
        list.append(word)


# returns list with every symplectic Hecke word of the same permutation as start_word of
↪  a fixed length by calculating the fitting equivalence class (see 2.2)
def HSp_list(start_word, length):
    liste=[]
    liste.append(start_word)
    for word in liste:
        # if any word in the list starts with an odd leter, start_word was not a
        ↪  symplectic Hecke word
        if word[0]%2==1:
            print('Error: This is not a symplectic Hecke word!')
            exit()
        # X(X+1)a ∼ X(X-1)a
        if 1<len(word) and abs(word[0]-word[1])==1 and word[1]+(2*(word[0]-word[1]))!=0:
            candidate = [word[0]]+[word[1]+(2*(word[0]-word[1]))]+word[2:len(word)]
            add_for_list(liste,candidate)
        for position in range(0,len(word)):
            # aXZXb ∼ aZXZb
            if position+2<len(word) and word[position]==word[position+2]:
                candidate = word[0:position] +
                ↪  [word[position+1],word[position],word[position+1]] +
                ↪  word[position+3:len(word)]
                add_for_list(liste,candidate)
            # aXYb ∼ aYXb if |X-Y|>1
            if position+1<len(word) and abs(word[position]-word[position+1])>1:
```

```
                candidate = word[0:position] + [word[position+1],word[position]] +
                ↪  word[position+2:len(word)]
                add_for_list(liste,candidate)
            # aXb -> aXXb
            if len(word)<length:
                candidate = word[0:position] + [word[position],word[position]] +
                ↪  word[position+1:len(word)]
                add_for_list(liste,candidate)
                if len(candidate)==length and len(start_word)!=length:
                    return HSp_list(candidate,length)
            # aXXb -> aXb
            if position+1<len(word) and word[position] == word[position+1]:
                candidate = word[0:position] + [word[position]] +
                ↪  word[position+2:len(word)]
                add_for_list(liste,candidate)
                if len(candidate)>length:
                    return HSp_list(candidate,length)
    # remove every member of the list, which has the wrong length
    kuerzer=[]
    for member in liste:
        if len(member)!=length:
            kuerzer.append(member)
    for member in kuerzer:
        liste.remove(member)
    return(liste)


# returns list with all FPF-involution words congruent to start_word (see 2.2)
def RFPF_list(start_word):
    list = [start_word]
    for word in list:
        # if any word in the list starts with an odd leter, start word was not a
        ↪  symplectic Hecke word
        if word[0]%2==1:
            print('Error: This is not a symplectic Hecke word!')
            exit()
        # X(X-1)a ∼ X(X+1)a
        if abs(word[0]-word[1])==1:
            add_for_list(list,[word[0],word[1]+(2*(word[0]-word[1]))] +
            ↪  word[2:len(word)])
            # if in the new word are same adjacent letters, start_word was not an
            ↪  FPF-involutions word
            if word[1]+(2*(word[0]-word[1])) == word[2]:
                return []
        # going through every letter of the actual word
        for position in range(0,len(word)-1):
            # aXYb ∼ aYXb for |X-Y| > 1
            if abs(word[position]-word[position+1]) != 1:
                candidate = word[0:position] + [word[position+1],word[position]] +
                ↪  word[position+2:len(word)]
                add_for_list(list,candidate)
                # if in the new word are same adjacent letters, start_word was not an
                ↪  FPF-involutions word
```

```
                if (position-1!=-1 and word[position-1] == word[position+1]) or
                ↪  (position != len(word)-2 and word[position]==word[position+2]):
                    return []
            if position != len(word)-2 and abs(word[position]-word[position+1]) == 1 and
            ↪  word[position]==word[position+2]:
                candidate = word[0:position] +
                ↪  [word[position+1],word[position],word[position+1]] +
                ↪  word[position+3:len(word)]
                add_for_list(list,candidate)
                # if in the new word are same adjacent letters, start_word was not an
                ↪  FPF-involutions word
                if word[position-1] == word[position+1] or word[position+1] ==
                ↪  word[position+2]:
                    return []
    return list


# returns ``symplectic product'' θ⊙w_1⊙w_2⊙... from word w, cutted at finite point
↪  (see 2.2)
def odot(word,function=()):
    s = [PermutationGroupElement([])]
    for i in range(1,2*ceil(max(word)/2)+2):
        s.append(PermutationGroupElement([(i,i+1)]))
    if function==():
        function=s[1]
        for i in range(3,2*ceil(max(word)/2)+2,2):
            function=function*s[i]
    for i in word:
        if function(i)<function(i+1):
            function = s[i]*function*s[i]
        else:
            if function(i)==i+1 and function(i+1)==i:
                return s[0]
    return function


# returns list with descents of a word or set-valued tableau (see 3.24)
def Des(input):
    descents=[]
    if type(input[0])!=list:
        word=input
        for index in range(0,len(word)-1):
            if word[index]>word[index+1]:
                descents.append(index+1)
        return descents
    else:
        Q=input
        if Q==[[[]]]:
            return []
        # double saves the row of unprimed entries and the column of primed ones (see
        ↪  remark 3.26)
        double={}
        for zeile in range(0,len(Q)):
            for spalte in range(0,len(Q[zeile])):
```

```
                for entry in range(0,len(Q[zeile][spalte])):
                    if Q[zeile][spalte][entry]==int(ceil(Q[zeile][spalte][entry])):
                        double[ceil(Q[zeile][spalte][entry])]=zeile
                    else:
                        double[ceil(Q[zeile][spalte][entry])]=spalte+zeile
        for index in range(1,len(double)):
            if double[index]<double[index+1]:
                descents.append(index)
    return descents


# returns true iff factorization (as list) is a valid weakly increasing factorization of
↪  word
def is_wif(word,factorization):
    # check, whether factorization is increasing
    sort=deepcopy(factorization)
    sort.sort()
    if factorization!=sort:
        return false
    # if word and factorization have different length
    if len(word)!=len(factorization):
        return false
    # if word decreases, but factorizations does not
    for index in Des(word):
        if factorization[index-1]>=factorization[index]:
            return false
    return true
```

5.2 Tableau methods

```
# checks, whether a shifted marked tableau is increasing in rows and columns
def is_increasing(tableau):
    for zeile in range(0,len(tableau)):
        for spalte in range(0,len(tableau[zeile])):
            # row is not the highest
            if zeile+1<len(tableau):
                # if entry over actual entry exists
                if spalte <= len(tableau[zeile+1]):
                    # and if this entry is smaller (we need column -1 to get relative
                    ↪  position)
                    if tableau[zeile][spalte] >= tableau[zeile+1][spalte-1]:
                        return false
            # if entry is not the last in its row
            if spalte+1 < len(tableau[zeile]):
                # and if the entry is geq than the one on the right
                if tableau[zeile][spalte] >= tableau[zeile][spalte+1]:
                    return false
    return true


# return list with the entries in the (absolute) index-th column of shifted tableau
def create_spalte(tableau,index):
    spalte=[]
```

```
    for zeile in range(0,len(tableau)):
        if index-zeile>=0 and index-zeile<len(tableau[zeile]):
            if type(tableau[zeile][index-zeile])!=list:
                spalte.append(tableau[zeile][index-zeile])
            else:
                spalte.append(tableau[zeile][index-zeile])
    return spalte


# returns the row reading word of a shifted insertion state
def row_word(tableau,outer_box=[None,None,None]):
    word=[]
    if outer_box[0]!=None and outer_box[1]==len(tableau):
        word.append(outer_box[0])
    for zeile in reversed(range(0,len(tableau))):
            word.extend(tableau[zeile])
            if outer_box[0]!=None and outer_box[1]==zeile:
                word.append(outer_box[0])
    return word


# returns the column reading word of a shifted insertion state
def column_word(tableau,outer_box=[None,None,None]):
    word=[]
    for index in range(0,len(tableau[0])):
        if outer_box[0]!=None and outer_box[2]==index:
            word.append(outer_box[0])
        word.extend(reversed(create_spalte(tableau,index)))
    if outer_box[0]!=None and outer_box[2]==len(tableau[0]):
        word.append(outer_box[0])
    return word


# retuns the word of a shifted insertion state
def word(tableau,outer_box=[None,None,None]):
    # if tableau has an outer column box
    if outer_box[1]==len(tableau)+1:
        word=column_word(tableau,outer_box)
    else:
        word=row_word(tableau,outer_box)
    return word


# checks, whether shifted insertion state is weakly admissible (see 3.4)
def is_weaklyadmissible(tableau,outer_box=[None,None,None]):
    # terminal and initial tableaus
    if outer_box[0]==None or outer_box[1]==0:
        return true
    # outer row box (first point of definition)
    if outer_box[2]==len(tableau[0])+1:
        # looking for fitting column
        for index in range(1,len(tableau[outer_box[1]-1])):
            # if row with outer box end before index-th entry
            if outer_box[1]==len(tableau) or index>len(tableau[outer_box[1]]):
```

```
                if tableau[outer_box[1]-1][index]<=outer_box[0]:
                    return true
            else:
                if tableau[outer_box[1]-1][index]<=outer_box[0] and
                ↪  outer_box[0]<tableau[outer_box[1]][index-1]:
                    return true
    # outer column box (second point of definition)
    if outer_box[1] == len(tableau)+1:
        # not allowed to have box in first column
        if outer_box[2] == 0:
            return false
        # checks for special condition
        if outer_box[2]-1<len(tableau) and
        ↪  outer_box[2]-(outer_box[2]-1)<len(tableau[outer_box[2]-1]):
            if tableau[outer_box[2]-1][outer_box[2]-(outer_box[2]-1)] == outer_box[0]:
                return true
        # collect columns of o.b. and the one before
        spalte_k = create_spalte(tableau,outer_box[2])
        spalte_j = create_spalte(tableau,outer_box[2]-1)
        # looking for fitting row
        for zeile in range(0,len(spalte_j)):
            # if the column beneath the o.b. ends before zeile-th entry
            if zeile>=len(spalte_k):
                if spalte_j[zeile]<=outer_box[0]:
                    return true
            else:
                if spalte_j[zeile]<=outer_box[0] and outer_box[0]<spalte_k[zeile]:
                    return true
    # wrong position for outer box
    return false


# checks, whether shifted insertion state is admissible (see 3.11)
def is_admissible(tableau,outer_box=[None,None,None]):
    if is_weaklyadmissible(tableau,outer_box) is false:
        return false
    # if word of tableau is not a symplectic Hecke word
    if odot(word(tableau,outer_box))==PermutationGroupElement([]):
        return false
    # if tableau is empty
    if tableau==[]:
        return true
    # if tableau has an outer row box
    if outer_box[2]==len(tableau[0])+1:
        # if (o.b.[1],o.b.[1]-1) is occupied by tableau
        if outer_box[1]<len(tableau) and 2<=len(tableau[outer_box[1]]):
            # if it has the value of the outer box but the diagonal box above is
            ↪  occupied
            if 1<len(tableau[outer_box[1]-1]) and outer_box[0] ==
            ↪  tableau[outer_box[1]-1][1] and outer_box[1]>=len(tableau[outer_box[1]]):
                return false
    # if tableau has an outer column box
    if outer_box[1]==len(tableau)+1:
        # if (o.b.[2]-1,o.b.[2]) is occupied by tableau
```

```
        if outer_box[2]-1<len(tableau) and outer_box[2]<len(tableau[outer_box[2]-1]):
            # if value of outer box is at position (o.b.[2]-1,o.b.[2]) in tableau
            if outer_box[0] == tableau[outer_box[2]-1][outer_box[2]]:
                # but (o.b.[2],o.b.[2]) isnt occupied and the outer box is even
                if outer_box[2]<len(tableau) and outer_box[2]<len(tableau[outer_box[2]])
                ↪  and outer_box[2]%2==0:
                    return false
        spalte_i=create_spalte(tableau,outer_box[2]-1)
        for index in range(0,len(spalte_i)):
            if outer_box[0]==spalte_i[index]:
                if index < 1:
                    return false
    return true


# returns standardisation of set-valued shifted tableau (see 3.28)
def standardization(tableau):
    spalten=[]
    for index in range(0,len(tableau[0])):
        spalten.append(create_spalte(tableau,index))
    std=deepcopy(tableau)
    maximum=0
    for zeile in range(0,len(tableau)):
        for spalte in range(0,len(tableau[zeile])):
            candidat=round(max(tableau[zeile][spalte]))
            if candidat > maximum:
                maximum=int(candidat)
    count=1
    for number in range(1,maximum+1):
        for spalte in range(0,len(tableau[0])):
            for zeile in range(0,len(spalten[spalte])):
                for entry in range(0,len(spalten[spalte][zeile])):
                    if number==spalten[spalte][zeile][entry]:
                        std[zeile][spalte-zeile][entry]=count
                        count+=1
        for zeile in range(0,len(tableau)):
            for spalte in range(0,len(tableau[zeile])):
                for entry in range(0,len(tableau[zeile][spalte])):
                    if tableau[zeile][spalte][entry]==number+0.5:
                        std[zeile][spalte][entry]=count-0.5
                        count+=1
    return std


# adds value to the aboslute position in the set-valued tableau Q
def add_to_Q(Q,value,position):
    if position[0]>len(Q) or position[1]>len(Q[0]):
        print('Warning: can not add position', position, 'to', Q,'!')
        return Q
    # add row
    if position[0] == len(Q):
        Q.append([[value]])
    else:
        # add a new box
```

```
        if position[1]-position[0] == len(Q[position[0]]):
            Q[position[0]].append([value])
        # add to a box
        else:
            Q[position[0]][position[1]-position[0]].append(value)
    return Q
```

5.3 Insertion algorithm

```
# inserts the first letter of the word to P and record it in Q
# documentation=true returns the edged in the path in prettyprint and the bumping path
# word_as_outer_box=true changes the output: (P,word) will be understood as insertion
↪  state and the algorithm tries to complete the insertion
def insert(P,Q,word,documentation=false, word_as_outer_box=false):
    # catch first R1:
    if P==[]:
        P=[[word[0]]]
        Q=[[[1]]]
        bumping_path=[(1,1)]
        return P,Q,word[1:len(word)]
    # starting routine:
    # either create the the outer_box for initial insertion state or take word as
    ↪  outer_box
    if word_as_outer_box==false:
        outer_box=[word[0],0,len(P[0])+1]
    else:
        outer_box=deepcopy(word)
        word=[]
    bumping_path=[]
    position=0
    # looking for maximal value in Q, in order to be able to add the next number to Q
    for zeile in range(0,len(Q)):
        for spalte in range(0,len(Q[zeile])):
            maximum=round(max(Q[zeile][spalte]))
            if maximum > position:
                position=int(maximum)
    if documentation:
        prettyprint(P,outer_box)
    while outer_box[0]!=None:
        # is outer box at valid position?
        if not ((outer_box[2]==len(P[0])+1 and outer_box[1] in range(0,len(P)+1)) or
        ↪  (outer_box[1]==len(P)+1 and outer_box[2] in range(0,len(P[0])+1))):
            print('Error: outer-box is not on a valid position: P=', P, 'outer-box=',
            ↪  outer_box)
            print('Warning: Picture maybe incorrect')
            exit()
        # if the outer box is a row box:
        if outer_box[2]==len(P[0])+1:
            # if the outer box is in a row above the tableau
            if outer_box[1]==len(P):
                P.append([])
            # if the outer box is maximal in its row, we can only get R1 und R2:
            if P[outer_box[1]]==[] or outer_box[0] >= max(P[outer_box[1]]):
```

```
        c = deepcopy(P)
        c[outer_box[1]].append(outer_box[0])
        # R1 (see R1):
        if is_increasing(c):
            P[outer_box[1]].append(outer_box[0])
            outer_box[0]=None
            bumping_path.append((outer_box[1]+1,
            ↪  len(P[outer_box[1]])+outer_box[1]))
            add_to_Q(Q,position+1,(outer_box[1],
            ↪  len(P[outer_box[1]])+outer_box[1]-1))
            if documentation:
                document('R1',P,outer_box)
            continue
        # R2 (see R2):
        else:
            outer_box[0]=None
            bumping_path.append((outer_box[1]+1,
            ↪  len(P[outer_box[1]])+outer_box[1]))
            spalte = create_spalte(P,len(P[outer_box[1]])+outer_box[1]-1)
            add_to_Q(Q,position+1,(len(spalte)-1,
            ↪  len(P[outer_box[1]])+outer_box[1]-1))
            # if we did not need the new row from the start of row transitions,
            ↪  delete it
            if P[-1]==[]:
                P.remove([])
            if documentation:
                document('R2',P,outer_box)
            continue
    # looking for minimal column x in row of outer box, s.t. ob < P(zeile,x)
    for x in range(0,len(P[outer_box[1]])):
        if outer_box[0]<P[outer_box[1]][x]:
            break
    # if the outer box is greater than the first box in the row:
    # R3 and R4 and D1:
    if x!=0:
        c = deepcopy(P)
        c[outer_box[1]][x] = outer_box[0]
        # R4 (see R4)
        if is_increasing(c):
            outer_box[0] = P[outer_box[1]][x]
            P[outer_box[1]][x] = c[outer_box[1]][x]
            outer_box[1] += 1
            bumping_path.append((outer_box[1],x+outer_box[1]))
            if documentation:
                document('R4',P,outer_box)
            continue
        else:
            # R3 (see R3)
            if x>1 or outer_box[1]<=len(P)-2:
                outer_box[0] = P[outer_box[1]][x]
                outer_box[1] += 1
                bumping_path.append((outer_box[1],x+outer_box[1]))
                if documentation:
                    document('R3',P,outer_box)
```

```
                continue
            # D1 (see D1)
            else:
                outer_box[0] = P[outer_box[1]][x]
                outer_box[2] = outer_box[1]+1
                outer_box[1] = len(P)+1
                bumping_path.append((outer_box[2],outer_box[2]+1))
                if documentation:
                    document('D1',P,outer_box)
                continue
    # else, s.t. the outer box is smaller or equal the first entry in its row:
    # D2, D3 and D4:
    else:
        # if diagonal entry and outer box have the same parity:
        # D2 and D3
        if P[outer_box[1]][0]%2==outer_box[0]%2:
            c = deepcopy(P)
            c[outer_box[1]][0] = outer_box[0]
            # D3 (see D3)
            if is_increasing(c):
                outer_box[0]=P[outer_box[1]][0]
                P[outer_box[1]][0]=c[outer_box[1]][0]
                outer_box[2]=outer_box[1]+1
                outer_box[1]=len(P)+1
                bumping_path.append((outer_box[2],outer_box[2]))
                if documentation:
                    document('D3',P,outer_box)
                continue
            # D2 (see D2)
            else:
                outer_box[0]=P[outer_box[1]][0]
                outer_box[2]=outer_box[1]+1
                outer_box[1]=len(P)+1
                bumping_path.append((outer_box[2],outer_box[2]))
                if documentation:
                    document('D2',P,outer_box)
                continue
        # if the diagonal entry and the outer box have different parities:
        # D4 (see D4)
        else:
            outer_box[0]=P[outer_box[1]][0]+1
            outer_box[2]=outer_box[1]+1
            outer_box[1]=len(P)+1
            bumping_path.append((outer_box[2],outer_box[2]))
            if documentation:
                document('D4',P,outer_box)
            continue
# else, s.t. if there is a column box
else:
    if outer_box[1]==len(P)+1:
        # take the column of the outer box from the tableau
        spalte=create_spalte(P,outer_box[2])
        # if outer box is maximal in its column:
        # C1 and C2:
```

```
            if spalte==[] or outer_box[0] >= max(spalte):
                # C1 (see C1)
                c = deepcopy(P)
                if len(spalte)==len(c):
                    c.append([outer_box[0]])
                else:
                    c[len(spalte)].append(outer_box[0])
                if is_increasing(c):
                    P=c
                    outer_box[0]=None
                    bumping_path.append((len(spalte)+1,outer_box[2]+1))
                    add_to_Q(Q,position+0.5,(len(spalte),outer_box[2]))
                    if documentation:
                        document('C1',P,outer_box)
                    continue
                # C2 (see C2)
                else:
                    outer_box[0]=None
                    bumping_path.append((len(spalte)+1,outer_box[2]+1))
                    add_to_Q(Q,position+0.5,(len(spalte)-1,
                    ↪  len(Q[len(spalte)-1])-1+len(spalte)-1))
                    if documentation:
                        document('C2',P,outer_box)
                    continue
            # if the outer box is not maximal in its column:
            # C3 and C4:
            # looking for minimal x in column i of the o.b., s.t. o.b.<P(i,x)
            for x in range(0,len(spalte)):
                if outer_box[0]<spalte[x]:
                    break
            c=deepcopy(P)
            c[x][outer_box[2]-x]=outer_box[0]
            # C4 (see C4)
            if is_increasing(c):
                outer_box[2]+=1
                outer_box[0]=P[x][outer_box[2]-x-1]
                P=c
                bumping_path.append((x+1,outer_box[2]-x))
                if documentation:
                    document('C4',P,outer_box)
                continue
            # C3 (see C3)
            else:
                outer_box[0]=P[x][outer_box[2]-x]
                outer_box[2]+=1
                bumping_path.append((x+1,outer_box[2]))
                if documentation:
                    document('C3',P,outer_box)
                continue
        print('Error: ended while-loop without declaring type of edge with')
        print('P=', P, 'outer box =', outer_box, 'and Q=', Q)
        exit()
    if documentation:
        print('and bumping path', bumping_path)
```

```
    return P,Q,word[1:len(word)]


# inserts the full input word in an empty tableau P and creates the recording tableau Q
# with input = (word,factorization) the algorithm returns the symplectic variant of Q
# documentation=true prints the insertion step by step, print_result= prettyprints the
↪  output
def sHi(input,documentation=false,print_result=false):
    if type(input)==tuple:
        word=input[0]
        factorization=input[1]
    else:
        word=input
        factorization=None
    # if factorization is not valid for word
    if factorization!=None and not(is_wif(word,factorization)):
        print('Error: this is not a valid weakly increasing factorization of the word!')
        exit()
    if print_result:
        print('The tableau resulting as insertion tableau when inserting', word, end='')
        if factorization!=None:
            print(' with weakly increasing factorization', factorization, end='')
        print(' to an empty tableau is')
    P=[]
    Q=[[]]
    # as long as we have letters in w, insert the next one
    while len(word)!=0:
        (P,Q,word)=insert(P,Q,word,documentation)
    # if a valid factorization was in input, return Q(w,i) (see 3.27)
    if factorization!=None:
        for zeile in range(0,len(Q)):
            for spalte in range(0,len(Q[zeile])):
                for entry in range(0,len(Q[zeile][spalte])):
                    if Q[zeile][spalte][entry]==ceil(Q[zeile][spalte][entry]):
                        Q[zeile][spalte][entry]=
                        ↪  factorization[ceil(Q[zeile][spalte][entry])-1]
                    else:
                        Q[zeile][spalte][entry]=
                        ↪  factorization[ceil(Q[zeile][spalte][entry])-1]-0.5
    if print_result:
        print('\n P=')
        prettyprint(P)
        print('\n with the recording tableau Q=')
        prettyprint(Q)
    return (P,Q)
```

5.4 Inverse insertion

```
# returns the bigger and existing entry in the tableau of the two absolute positions
def bigger_entry(tableau,position1,position2):
    # if position1 is occupied by tableau
    if position1[0]<len(tableau) and position1[1]-position1[0]<
    ↪  len(tableau[position1[0]]) and position1[0]>=0 and position1[1]>=0:
```

```
        candidate=tableau[position1[0]][position1[1]-position1[0]]
        # if position2 is also occupied by tableau
        if position2[0]<len(tableau) and position2[1]-position2[0]<
        ↪  len(tableau[position2[0]]) and position2[0]>=0 and position2[1]>=0:
            candidate=max(tableau[position1[0]][position1[1]-position1[0]],
            ↪  tableau[position2[0]][position2[1]-position2[0]])
    # if position1 is not occupied by tableau
    else:
        # if position2 is occupied by tableau
        if position2[0]<len(tableau) and position2[1]-position2[1]<
        ↪  len(tableau[position2[0]]) and position2[0]>=0 and position2[1]>=0:
            candidate=tableau[position2[0]][position2[1]-position2[0]]
        else:
            print('\n ERROR: Neither', position1, 'nor', position2, 'is occupied by',
            ↪  tableau)
            exit()
    return candidate


# removes maximal value from box in relative position
def remove_from_Q(Q,pos_zeile,pos_spalte):
    if len(Q[pos_zeile][pos_spalte])!=1:
        Q[pos_zeile][pos_spalte].remove(max(Q[pos_zeile][pos_spalte]))
    else:
        if len(Q[pos_zeile])!=1:
            Q[pos_zeile].remove([max(Q[pos_zeile][pos_spalte])])
        else:
            Q.remove([[max(Q[pos_zeile][pos_spalte])]])


# uninserts the value of P, which is choosen by Q and put it in front of the word
def uninsert(P,Q,word,documentation=false):
    if documentation:
        prettyprint(P)
    # looking for maximal element in Q and save its relative position in (pos_zeile,
    ↪  pos_spalte)
    position=0
    for zeile in range(0,len(Q)):
        for spalte in range(0,len(Q[zeile])):
            maximum=round(max(Q[zeile][spalte]))
            if maximum > position:
                position=int(maximum)
                pos_zeile=zeile
                pos_spalte=spalte
    outer_box=[None,None,None]
    # if maximal element is unprimed:
    # iR1 or iR2
    if max(Q[pos_zeile][pos_spalte])==position:
        # if it is alone in its box, so iR1 (see iR1)
        if len(Q[pos_zeile][pos_spalte])==1:
            outer_box[0]=P[pos_zeile][pos_spalte]
            outer_box[1]=pos_zeile
            outer_box[2]=len(P[0])+1
            if len(P[pos_zeile])!=1:
```

```
            P[pos_zeile].pop(pos_spalte)
        else:
            P.pop(pos_zeile)
        if documentation:
            document('iR1',P,outer_box)
    # else, s.t. there are other elements in the box of the maximal value in Q, so
    ↪  iR2 (see iR2)
    else:
        # take the right outer corner
        outer_corner=[None,None,pos_spalte+1+pos_zeile]
        if pos_spalte+1<len(P[0]):
            outer_corner[1]=len(create_spalte(P,pos_spalte+pos_zeile+1))
        else:
            outer_corner[1]=0
        # now iR2 at position of the outer corner
        outer_box[1]=outer_corner[1]
        outer_box[2]=len(P[0])+1
        outer_box[0]=bigger_entry(P,(outer_corner[1],outer_corner[2]-1),
        ↪  (outer_corner[1]-1,outer_corner[2]))
        if documentation:
            document('iR2',P,outer_box)
# else, s.t. maximal value in Q is primed, so
# iC1 and iC2
else:
    # if it is alone in its box, so iC1 (see iC1)
    if len(Q[pos_zeile][pos_spalte])==1:
        outer_box[0]=P[pos_zeile][pos_spalte]
        outer_box[1]=len(P)+1
        outer_box[2]=pos_spalte+pos_zeile
        if len(P[pos_zeile])!=1:
            P[pos_zeile].pop(pos_spalte)
        else:
            P.pop(pos_zeile)
        if documentation:
            document('iC1',P,outer_box)
    # else, s.t. there are other elements in the box of the maximal value in Q, so
    ↪  iC2 (see iC2)
    else:
        # looking for outer corner in P in row above of max value in Q
        outer_corner=[None,pos_zeile+1,None]
        if pos_zeile+1<len(P):
            outer_corner[2]=len(P[pos_zeile+1])+pos_zeile+1
        else:
            outer_corner[2]=pos_zeile+1
        # now iC2 at position of outer corner
        outer_box[2]=outer_corner[2]
        outer_box[1]=len(P)+1
        outer_box[0]=bigger_entry(P,(outer_corner[1]-1,outer_corner[2]),
        ↪  (outer_corner[1],outer_corner[2]-1))
        if documentation:
            document('iC2',P,outer_box)
if outer_box[0]==None or outer_box[1]==None or outer_box[2]==None:
    print('\n Error: Couldnt find the first edge, P=', P, 'outer_box=', outer_box)
    exit()
```

```
if is_admissible(P,outer_box) is false:
    print('\n Error: Something went wrong, P is not admissible anymore! \n P=', P,
    ↪  'outer_box=', outer_box)
    prettyprint(P, outer_box)
    exit()
# other edges:
# starting routine
while outer_box[1]!=0:
    if is_admissible(P,outer_box) is false:
        print('\n Error: Something went wrong, P is not admissible anymore! \n
        ↪  outer_box=', outer_box)
        prettyprint(P)
        exit()
    # if outer box is an outer row box
    if outer_box[2]==len(P[0])+1:
        # looking for special column from admissibility
        for spalte in reversed(range(0,len(P[outer_box[1]-1]))):
            if P[outer_box[1]-1][spalte]<=outer_box[0]:
                break
        # iR3 (see iR3)
        if P[outer_box[1]-1][spalte]==outer_box[0]:
            outer_box[1]-=1
            outer_box[0]=bigger_entry(P,(outer_box[1],spalte-1-outer_box[1]),
            ↪  (outer_box[1]-1,spalte+1-(outer_box[1]-1)))
            if documentation:
                document('iR3',P,outer_box)
            continue
        # iR4: (see iR4)
        else:
            outer_box[1]-=1
            outer_box[0], P[outer_box[1]][spalte] = P[outer_box[1]][spalte],
            ↪  outer_box[0]
            if documentation:
                document('iR4',P,outer_box)
            continue
    # else, s.t if the outer box is an outer column box
    else:
        if outer_box[1]==len(P)+1:
            # if value of outer box is geq than diagonal entry in column left of
            ↪  outer box:
            # iD1 or iD2 or iD3 or iD4 or iC3a
            if outer_box[2]-1<len(P) and P[outer_box[2]-1][0]<=outer_box[0]:
                # iD1: (see iD1)
                if outer_box[0]%2==0 and P[outer_box[2]-1][0]<outer_box[0] and
                ↪  (outer_box[2]<len(P[outer_box[2]-1]) and
                ↪  P[outer_box[2]-1][1]==outer_box[0]):
                    outer_box[0]=bigger_entry(P,(outer_box[2]-1,0),
                    ↪  (outer_box[2]-2,2))
                    outer_box[1]=outer_box[2]-1
                    outer_box[2]=len(P[0])+1
                    if documentation:
                        document('iD1',P,outer_box)
                    continue
                # iD2 (see iD2)
```

```
    if P[outer_box[2]-1][0]==outer_box[0] and P[outer_box[2]-2][1]%2==0:
        outer_box[0]=P[outer_box[2]-2][1]
        outer_box[1]=outer_box[2]-1
        outer_box[2]=len(P[0])+1
        if documentation:
            document('iD2',P,outer_box)
        continue
    # iD3 (see iD3)
    if P[outer_box[2]-1][0]<outer_box[0] and (1>=len(P[outer_box[2]-1])
    ↪  or outer_box[0]<P[outer_box[2]-1][1]) and outer_box[0]%2==0:
        outer_box[1]=outer_box[2]-1
        outer_box[2]=len(P[0])+1
        outer_box[0],P[outer_box[1]][0] =
        ↪  P[outer_box[1]][0],outer_box[0]
        if documentation:
            document('iD3',P,outer_box)
        continue
    # iD4 (see iD4)
    if P[outer_box[2]-1][0]<outer_box[0] and outer_box[0]%2==1:
        outer_box[0]=P[outer_box[2]-1][0]-1
        outer_box[1]=outer_box[2]-1
        outer_box[2]=len(P[0])+1
        if documentation:
            document('iD4',P,outer_box)
        continue
    # iC3a (see iC3a)
    if P[outer_box[2]-1][0]==outer_box[0] and 1<len(P[outer_box[2]-2])
    ↪  and P[outer_box[2]-2][1]%2==1:
                outer_box[0]=P[outer_box[2]-2][1]
                outer_box[2]-=1
                if documentation:
                    document('iC3a',P,outer_box)
                continue
# if outer box is lower than the diagonal entry in column left of outer
↪  box:
# iC3b and iC4
if outer_box[2]-1>=len(P) or outer_box[0]<P[outer_box[2]-1][0]:
    # looking for special row given by admissible tableaus
    spalte=create_spalte(P,outer_box[2]-1)
    for index in reversed(range(0,len(spalte))):
        if spalte[index]<=outer_box[0]:
            break
    # iC3b (see iC3b)
    if P[index][outer_box[2]-1-index]==outer_box[0]:
        outer_box[2]-=1
        outer_box[0]=bigger_entry(P,(index-1,outer_box[2]),
        ↪  (index,outer_box[2]-1))
        if documentation:
            document('iC3b',P,outer_box)
        continue
    # iC4 (see iC4)
    if P[index][outer_box[2]-1-index]<outer_box[0]:
        outer_box[2]-=1
```

```
                        outer_box[0],P[index][outer_box[2]-index] =
                        ↪  P[index][outer_box[2]-index],outer_box[0]
                        if documentation:
                            document('iC4',P,outer_box)
                        continue
            print('\n Error: Something went wrong. Can not find fitting edge for P=')
            prettyprint(P,outer_box)
            exit()
        # remove max element from Q and add outer box to word
        remove_from_Q(Q,pos_zeile,pos_spalte)
        word.insert(0, outer_box[0])
        return(P,Q,word)


# returns the word (resp. the word and the factorization), which results in the
↪  admissible insertion tableau P and the recording tableau Q by symplectic Hecke
↪  insertion.
def isHi(PQ,documentation=false,print_result=false):
    P=PQ[0]
    Q=PQ[1]
    # looking for factorization:
    factorization=[]
    for zeile in range(0,len(Q)):
        for spalte in range(0,len(Q[zeile])):
            for entry in range(0,len(Q[zeile][spalte])):
                factorization.append(ceil(Q[zeile][spalte][entry]))
    factorization.sort()
    # if factorization is not trivial, calculate standardization(Q(w,i))
    if factorization!=list(range(1,len(factorization)+1)):
        Q=standardization(Q)
    if print_result:
        print('To get this P and Q you need the word ', end='')
    word=[]
    # uninsert letters from P, till it is empty
    while P!=[]:
        (P,Q,word)=uninsert(P,Q,word,documentation)
    if print_result:
        print(word, end='')
        if factorization!=list(range(1,len(factorization)+1)):
            print(' with the factorization', factorization,end='')
        print('.')
    if factorization == list(range(1,len(factorization)+1)):
        return word
    return (word,factorization)
```

5.5 Prettyprinting and documentation

```
# prints row with entries
def printentries(tableau,zeile,spalte):
    if type(tableau[zeile][spalte])==type([]):
        for eintrag in range(0,len(tableau[zeile][spalte])):
            if type(tableau[zeile][spalte][eintrag])==Integer or
            ↪  type(tableau[zeile][spalte][eintrag]) == int:
```

```
                print(tableau[zeile][spalte][eintrag], sep='',end=' ')
            else:
                print(int(tableau[zeile][spalte][eintrag]+0.5),'´', sep='',end=' ')
    else:
            print(tableau[zeile][spalte], sep='',end=' ')
# create tableau, which entries save the needed length per box
def create_length(tableau):
    laenge=[]
    # Set-valued tableaus:
    if type(tableau[0][0])==type([]):
        for zeile in range(0,len(tableau)):
            laenge.append([])
            for spalte in range(0,len(tableau[zeile])):
                laenge[zeile].append([])
                laenge[zeile][spalte]=2*len(tableau[zeile][spalte])-1
                for eintrag in range(0,len(tableau[zeile][spalte])):
                    laenge[zeile][spalte]+=
                    ↪  floor(log(tableau[zeile][spalte][eintrag]+0.5,10))
                    if round(tableau[zeile][spalte][eintrag])!=
                    ↪  tableau[zeile][spalte][eintrag]:
                        laenge[zeile][spalte]+=1
    # insertion tableaus
    else:
        for zeile in range(0,len(tableau)):
            laenge.append([])
            for spalte in range(0,len(tableau[zeile])):
                laenge[zeile].append([])
                laenge[zeile][spalte] = 1+floor(log(tableau[zeile][spalte]+0.5,10))
    return laenge


# print (set-valued) shifted tableaus (with outer box):
def prettyprint(tableau, outer_box=[None,None,None]):
    # catch empty tableaus
    if (tableau==[] or tableau==[[]]) and outer_box[0]==None:
        print('######')
        print('#')
        print('#')
        print('######')
        return
    if (tableau==[] or tableau==[[]]) and outer_box[0]!=None:
        print('######')
        print('#      +-','-'*(1+floor(log(outer_box[0]+0.5,10))),'-+',sep='')
        print('#      | ',outer_box[0],' |',sep='')
        print('#      +-','-'*(1+floor(log(outer_box[0]+0.5,10))),'-+',sep='')
        print('######')
        return
    # first determine the length needed for each box
    laenge=create_length(tableau)
    # looking for maximal length in each column to make every box in a column equaly
    ↪  long
    laenge_spalte=deepcopy(laenge[0])
    # looking for outer column box
    if outer_box[0]!=None and outer_box[1]==len(tableau)+1:
```

```
    if outer_box[2]<len(laenge_spalte):
        laenge_spalte[outer_box[2]]=
        ↪  max(laenge_spalte[outer_box[2]],1+floor(log(outer_box[0]+0.5,10)))
if len(laenge)>0:
    for zeile in range(1,len(laenge)):
        for spalte in range(0,len(laenge[zeile])):
            laenge_spalte[zeile+spalte]=
            ↪  max(laenge_spalte[zeile+spalte],laenge[zeile][spalte])
# create frame
print()
print('###','#'*(sum(laenge_spalte)+3*len(tableau[0])),sep='',end='')
if outer_box[0]!=None:
    print('###'*(max(0,outer_box[2]-len(tableau[0]))),
    ↪  '#'*(1+floor(log(outer_box[0]+0.5,10))), '###', sep='',end='')
print('###')
print('#')
# create boxes with entries
# create outer column box
if outer_box[0]!=None and outer_box[1]==len(tableau)+1 and
↪  outer_box[2]<len(laenge_spalte):
    print('#  ', '    '*outer_box[2], '
    ↪  '*(sum(laenge_spalte[0:outer_box[2]])-outer_box[1]), sep='',end='')
    print('+-', '-'*laenge_spalte[outer_box[2]],'-+', sep='')
    print('#  ', '    '*outer_box[2], '
    ↪  '*(sum(laenge_spalte[0:outer_box[2]])-outer_box[1]), sep='',end='')
    print('|', outer_box[0],'|')
    print('#  ','    '*outer_box[2], '
    ↪  '*(sum(laenge_spalte[0:outer_box[2]])-outer_box[1]), sep='',end='')
    print('+-','-'*laenge_spalte[outer_box[2]], '-+', sep='')
    print('#  ')
else:
    # create outer column box in a column next to tableau
    if outer_box[0]!=None and outer_box[1]==len(tableau)+1:
        print('#  ', '   '*(len(tableau[0])), ' '*(sum(laenge_spalte)+1),
        ↪  sep='',end='')
        print('+-', '-'*(1+floor(log(outer_box[0]+0.5,10))),'-+',sep='')
        print('#  ', '   '*(len(tableau[0])), ' '*(sum(laenge_spalte)+1),
        ↪  sep='',end='')
        print('|', outer_box[0],'|')
        print('#  ', '   '*(len(tableau[0])), ' '*(sum(laenge_spalte)+1),
        ↪  sep='',end='')
        print('+-', '-'*(1+floor(log(outer_box[0]+0.5,10))), '-+', sep='')
        print('#  ')
# create outer row box in row over tableau
if outer_box[0]!=None and outer_box[1]==len(tableau) and
↪  outer_box[2]==len(tableau[0])+1:
    print('#  ', '   '*(len(tableau[0])+1), '
    ↪  '*(sum(laenge_spalte)+1),sep='',end='')
    print('+-', '-'*(1+floor(log(outer_box[0]+0.5,10))),'-+',sep='')
    print('#  ', '   '*(len(tableau[0])+1), '
    ↪  '*(sum(laenge_spalte)+1),sep='',end='')
    print('|', outer_box[0],'|')
print('#  ', end='')
# create tableau
```

```
    for zeile in reversed(range(0,len(laenge))):
        # Shifting
        print('    '*zeile,' '*(sum(laenge_spalte[0:zeile])-zeile), sep='',end='')
        # edge above row
        for spalte in range(0,len(laenge[zeile])):
            print('+-','-'*laenge_spalte[spalte+zeile],'-', sep='',end='')
        print('+',end='')
        # edges of outer row box
        if outer_box[0]!=None and (outer_box[1]==zeile or outer_box[1]==zeile+1):
            print(' '*(sum(laenge_spalte[len(tableau[zeile])+zeile:len(tableau[0])])), '
            ↪  '*(len(tableau[0])-len(tableau[zeile])-zeile+1), sep='', end='')
            print('+-','-'*(1+floor(log(outer_box[0]+0.5,10))),'-+', sep='',end='')
        print()
        print('#  ',end='')
        # shifting
        print('    '*zeile,' '*(sum(laenge_spalte[0:zeile])-zeile), sep='',end='')
        # print entries
        for spalte in range(0,len(laenge[zeile])):
                print('| ', '
                ↪  '*floor((laenge_spalte[spalte+zeile]-laenge[zeile][spalte])/2),
                ↪  sep='',end='')
                printentries(tableau,zeile,spalte)
                print(' '*ceil((laenge_spalte[spalte+zeile]-laenge[zeile][spalte])/2),
                ↪  sep='',end='')
        print('| ',end='')
        # print entry of outer row box
        if outer_box[0]!=None and outer_box[1]==zeile:
            print(' '*(sum(laenge_spalte[len(tableau[zeile])+zeile:len(tableau[0])])), '
            ↪  '*(len(tableau[0])-len(tableau[zeile])-zeile), sep='', end='')
            print('  |', outer_box[0],'|', end='')
        print()
        print('#  ',end='')
    # print lowest edge
    for spalte in range(0,len(laenge[0])):
        print('+-', '-'*laenge_spalte[spalte], '-', sep='',end='')
    print('+', end='')
    if outer_box[0]!=None and outer_box[1]==0:
        print('   +-', '-'*(1+floor(log(outer_box[0]+0.5,10))), '-+', sep='', end='')
    # create frame
    print()
    print('#')
    print('###','#'*(sum(laenge_spalte)+3*len(tableau[0])),sep='',end='')
    if outer_box[0]!=None:
        print('###'*(max(0,outer_box[2]-len(tableau[0]))),
        ↪  '#'*(1+floor(log(outer_box[0]+0.5,10))), '###', sep='', end='')
    print('###')
    print()


# print an arrow for documentation
def print_edge(label):
    print('  |')
    print('',label)
    print('  |')
```

```
    print('  V')


# prints an edge with hit tableau for documentation
def document(label,P,outer_box):
    print_edge(label)
    prettyprint(P,outer_box)
```

5.6 Examples

Finally, here are are some examples, which produce the output beneath them.

Input:

```
print('\n---------- \n' )

#In Example 1 we calculated, that
print('26542 is a symplectic Hecke word for')

odot([2,6,5,4,2])


print('\n---------- \n' )

#In example 4 we calculated, that
print('The FPF-involution words of (56)(67)(23)theta(23)(67)(56) are')

RFPF_list([2,6,5])


print('\n---------- \n' )

#In example 18 we saw the insertion tableau, in example 26 the recording tableau of
sHi([2,6,5,4,2,5], print_result=true)


print('\n---------- \n' )

#In exaple 27 we saw the inverse:
isHi(([[2, 4, 5, 7], [6]], [[[1], [2], [3.5], [4.5]], [[3, 6]]]))


print('\n---------- \n' )

#In example 30 we saw which recording tableau we get with a factorization:
sHi(([2,6,5,4,2,5],[1,1,2,3,4,4]),print_result=true)


print('\n---------- \n' )

#In example 33 we saw the insertion tableau of 265, here is the insertion step by step:
sHi([2,6,5],documentation=true)


print('\n---------- \n' )
```

Output:

```
----------

26542 is a symplectic Hecke word for
(1,3)(2,5)(4,8)(6,7)

----------

The FPF-involution words of (56)(67)(23)theta(23)(67)(56) are
[[2, 6, 5], [6, 2, 5], [6, 5, 2], [6, 7, 2], [6, 2, 7], [2, 6, 7]]

----------

The tableau resulting as insertion tableau when inserting [2, 6, 5, 4, 2, 5] to an empty
↪  tableau is

 P=

######################
#
#      +---+
#      | 6 |
#  +---+---+---+---+
#  | 2 | 4 | 5 | 7 |
#  +---+---+---+---+
#
######################


 with the recording tableau Q=

##########################
#
#      +-----+
#      | 3 6 |
#  +---+-----+----+----+
#  | 1 |  2  | 4´ | 5´ |
#  +---+-----+----+----+
#
##########################

([[2, 4, 5, 7], [6]], [[[1], [2], [3.50000000000000], [4.50000000000000]], [[3, 6]]])

----------

To get this P and Q you need the word [2, 6, 5, 4, 2, 5].
[2, 6, 5, 4, 2, 5]

----------

The tableau resulting as insertion tableau when inserting [2, 6, 5, 4, 2, 5] with weakly
↪  increasing factorization [1, 1, 2, 3, 4, 4] to an empty tableau is
```

```
 P=

######################
#
#      +---+
#      | 6 |
#  +---+---+---+---+
#  | 2 | 4 | 5 | 7 |
#  +---+---+---+---+
#
######################


 with the recording tableau Q=

##########################
#
#      +-----+
#      | 2 4 |
#  +---+-----+----+----+
#  | 1 |  1  | 3´ | 4´ |
#  +---+-----+----+----+
#
##########################

([[2, 4, 5, 7], [6]], [[[1], [1], [2.500000000000000], [3.500000000000000]], [[2, 4]]])

----------


#################
#
#  +---+   +---+
#  | 2 |   | 6 |
#  +---+   +---+
#
#################

  |
 R1
  |
  V

##############
#
#  +---+---+
#  | 2 | 6 |
#  +---+---+
#
##############

and bumping path [(1, 2)]

####################
```

```
#
#  +---+---+   +---+
#  | 2 | 6 |   | 5 |
#  +---+---+   +---+
#
######################

  |
 R4
  |
  V

######################
#
#              +---+
#              | 6 |
#  +---+---+   +---+
#  | 2 | 5 |
#  +---+---+
#
######################

  |
 R1
  |
  V

###############
#
#      +---+
#      | 6 |
#  +---+---+
#  | 2 | 5 |
#  +---+---+
#
###############

and bumping path [(1, 2), (2, 2)]
([[2, 5], [6]], [[[1], [2]], [[3]]])

----------
```

References

[1] T. Ikeda and H. Naruse. "K-theoretic analogues of factorial Schur P- and Q-functions", Advances in Mathematics 243 (2013), pp. 22–66.

[2] E. Clifford, H. Thomas and A. Yong. "K-theoretic Schubert calculus for OG(n,2n+1) and jeu de taquin for shifted increasing tableaux", J. Reine Angew. Math. 690 (2014), pp. 51-63.

[3] E. Marberg and B. Pawlowski. "On some properties of symplectic Grothendieck polynomials", J. Pure Appl. Algebra 225 (2021), 106463.

[4] E. Marberg. "Shifted insertion algorithms for primed words", arXiv:2104.11437.

[5] E. Marberg. "A symplectic refinement of shifted Hecke insertion", J. Combin. Theory Ser. A 173 (2020), 105216.

[6] J. B. Lewis and E. Marberg. "Enriched set-valued P-partitions and shifted stable Grothendieck polynomials", Math. Z. 299 (2021), pp. 1929-1972.

Eigenständigkeitserklärung

Hiermit versichere ich, dass

1. ich die vorliegende Arbeit selbständig angefertigt habe,
2. ich außer den im Quellen- und Literaturverzeichnis sowie in den Anmerkungen genannten Hilfsmitteln keine weiteren benutzt,
3. ich alle Stellen der Arbeit, die anderen Werken dem Wortlaut oder Sinn nach entnommen sind, unter Angabe der Quelle als Entlehnung kenntlich gemacht habe. Das umfasst alle Quellen, insbesondere auch Informationen aus dem internet und
4. die vorliegende Arbeit mit der vorher abgegebenen Version übereinstimmt.

Gleichzeitig erkläre ich, dass ich weder diese Arbeit (in dieser oder einer inhaltlich äquivalenten Form) noch Teile daraus bereits an anderer Stelle eingereicht habe.

Datum, Ort

Unterschrift