TECHNISCHE UNIVERSITÄT WIEN
Vienna | Austria

Dissertation

A Sundaram type bijection for $\mathrm{SO}(2k+1)$: vacillating tableaux and pairs consisting of a standard Young tableau and an orthogonal Littlewood-Richardson tableau

Ausgeführt zum Zwecke der Erlangung des akademischen Grades
eines
Doktors der technischen Wissenschaften
unter der Leitung von

Privatdoz. Dr.rer.nat. Martin Rubey

Institut für Diskrete Mathematik und Geometrie E104
eingereicht an der Technischen Universität Wien
Fakultät für Mathematik und Geoinformation

von

Judith Jagenteufel
0926771
Friedhofstraße 1c/4/5
2103 Langenzersdorf

Wien, am

Unterschrift

Kurzfassung

In dieser Arbeit wird eine Bijektion zwischen Vacillating Tableaux und Paaren bestehend aus einem Standard Young Tableau und einem orthogonalen Littlewood-Richardson Tableau für die spezielle orthogonale Gruppe $SO(2k+1)$ in ungerader Dimension präsentiert. Diese ist motiviert durch die Zerlegung in direkte Summen des $r$ten Tensorprodukts der definierenden Darstellung von $SO(2k+1)$ und beantwortet eine in den späten Neuzehnachzigerjahren gestellte Frage von Sheila Sundaram.

Um die Bijektion zu beschreiben, wird zunächst eine explizite Formulierung von Kwon's Littlewood-Richardson Tableaux gefunden. Anschließend werden alternative Tableaux definiert und gezeigt, dass diese mit jenen von Kwon in Bijektion sind. Dafür wird eine explizite Bijektion angegeben. Mit Hilfe der alternativen Tableaux kann die ursprüngliche Fragestellung auf einen Spezialfall reduziert werden und eine weitere Bijektion gefunden werden, die ein Paar bestehend aus einem Standard Young Tableau und einem alternativen Tableau auf ein Vacillating Tableau mit gewünschten Eigenschaften abbildet.

Für $SO(3)$ wird eine alternative Formulierung der Bijektion präsentiert, die weitere Eigenschaften zeigt, welche in allgemeiner ungerader Dimension lediglich vermuten werden können. Auf der kombinatorischen Seite erhält man zudem eine Bijektion zwischen Riordanschen Pfaden und Standard Young Tableaux mit höchstens drei Zeilen, deren Längen die selbe Parität haben.

Außerdem wird eine passend definierte Menge von Descents für Vacillating Tableaux verwendet um die quasisymmetrische Zerlegung des Frobenius Charakters der isotypischen Komponenten zu erhalten.

Abstract

We present a bijection between vacillating tableaux and pairs consisting of a standard Young tableau and an orthogonal Littlewood-Richardson tableau for the special orthogonal group $\mathrm{SO}(2k+1)$ in odd dimension. This bijection is motivated by the direct-sum-decomposition of the rth tensor power of the defining representation of $\mathrm{SO}(2k+1)$. The question of such a bijection was first asked by Sheila Sundaram in the late nineteen-eighties and remained an open question since then.

To formulate it, we present an explicit formulation of Kwon's Littlewood-Richardson tableaux and define alternative tableaux. We show that our new alternative tableaux are in bijection with Kwon's tableaux by giving an explicit bijection. Those new tableaux help us to reduce the problem to a special case. We then define another bijection, which maps a pair consisting of a standard Young tableau and an alternative tableau to a vacillating tableau with desired properties.

For $\mathrm{SO}(3)$ we present an alternative formulation of our bijection, which proves further properties, that can only be conjectured for general odd dimension. On the combinatorial side we obtain a bijection between Riordan paths and standard Young tableaux with at most 3 rows, all of even length or all of odd length.

Moreover we use a suitably defined descent set for vacillating tableaux to determine the quasi-symmetric expansion of the Frobenius characters of the isotypic components.

Acknowledgements

It had been a long way from being a first-year-math-student to finally finishing my PhD. It had been a long way, but also a good one and there were a lot of people that accompanied me. Therefore I would like to thank every one who made my studies possible and the time being a great time.

In particular I want to thank:

- Martin Rubey for showing me the beauty of combinatorics, for introducing new problems like they were riddles to solve, for advising my PhD- and Masterthesis and for all the discussions we had in the previous years.
- Olga Azenhas and Christian Krattenthaler for reviewing my thesis and being part of my examination committee.
- Sheila Sundaram, for her ongoing interest in my work, for her helpful comments and for writing a thesis, that inspired mine.

Moreover I would like to thank all recent and past members of the research group at TU Wien in particular:

- Stephan Pfannerer for being my "PhD-brother", for valuable discussions and sharing the office.
- Felix Dellinger, Thomas Hack, Georg Hofstätter, Andreas Kreumel, Philip Kniefacz, Isabella Larcher, Olaf Mordhorst, Fabian Mußnig and Johannes Schürtz for lunch breaks, jokes and cheering me up.
- Edith Rosta for all her help and all the every-day-conversations that made my day a better one.

I also want to thank the Austrian state for the opportunity to study, the Austrian Science Found FWF for funding me and the TU Wien for providing such good opportunities to study and research.

Furthermore I want to thank my colleagues from my studies who share their passion for mathematics:

- Christina Satzinger for all those hours we spent in various lectures sitting next to each other, and for simply being my best friend.
- Clemens Müllner for all the times he explained something complicated to me.
- Christina Bohrn, Anita Gerstenmayer, Katharina Ölsböck and Elisabeth Schiessler for making being a girl at the TU Wien something normal.

Finally I want to thank my family:

- Sigi and Geri Braunsteiner for being my parents, for all their support and encouragement and their never ending love.
- Hannah and Tobi for being the best siblings I could wish for.
- Paul for sharing this live with me.

Contents

Introduction

We present a bijection for SO$(2k+1)$ between vacillating tableaux and pairs consisting of a standard Young tableau and an orthogonal Littlewood-Richardson tableau (introduced by Kwon in [5]). This bijection explains the direct-sum-decomposition of a tensor power $V^{\otimes r}$ of the defining representation V of SO$(2k+1)$ combinatorially. In particular we consider

$$V^{\otimes r} = \bigoplus_{\mu} V(\mu) \otimes U(r,\mu) = \bigoplus_{\mu} V(\mu) \otimes \bigoplus_{\lambda} c_{\lambda}^{\mu} S(\lambda)$$

as an SO$(2k+1) \times \mathfrak{S}_r$ representation. $V(\mu)$ is an irreducible representation of SO$(2k+1)$ and $S(\lambda)$ is a Specht module. We concentrate on $U(r,\mu)$. A basis of $U(r,\mu)$ can be indexed by vacillating tableaux. The multiplicities c_{λ}^{μ} can be obtained by counting orthogonal Littlewood-Richardson tableaux. A basis of $S(\lambda)$ is indexed by standard Young tableaux.

To formulate our bijection, we introduce an explicit combinatorial description of Kwon's orthogonal Littlewood-Richardson tableaux. Those are defined in a very general way in terms of crystal graphs. Using our new description we find an alternative set of orthogonal Littlewood-Richardson tableaux, which is in bijection with Kwon's set. We describe this bijection in Bijection A using Algorithm 1. Our alternative tableaux are described in terms of skew semistandard tableaux with a reading word that is Yamanouchi. Those are similar to Sundarams symplectic tableaux [13]. However, the additional condition we have is far more complicated than the one she obtained. This alternative set of tableaux reduces the problem to finding a bijection between vacillating tableaux and standard Young tableaux with $2k+1$ rows, all of them with lengths of the same parity. We solve this reduced problem with Bijection B described by Algorithm 3.

The question of finding such a bijection was posed by Sundaram in her 1986 thesis [13] and has been attacked several times since Sundaram's thesis; in particular by Sundaram [14] and Proctor [8]. Okada [7] recently obtained the decomposition of $U(r,\mu)$ for multiplicity free cases implicitly using representation theoretic computations. We obtain parts of these results as a special case, which are on their part special cases of Okada's work. In fact, Okada asks for bijective proofs of his results.

One might assume that Fomin's machinery of growth diagrams could be employed to find such a bijection. For the symplectic group this was done by Roby [9] and Krattenthaler [4]. However, for the orthogonal group the situation appears to be quite different. In particular, at least a naive application of Fomin's ideas does not even yield the desired bijection between vacillating tableaux and the set of standard Young tableaux in question, not even for dimension 3.

An advantage of our combinatorial, bijective approach is that we obtain additional properties and consequences such as the following.

We define a suitable notion of descents for vacillating tableaux and use the classical descent set for standard Young tableaux introduced by Schützenberger. We can show that our bijection is descent preserving. Thus we obtain the quasi-symmetric expansion of the Frobenius character of the isotypic space $U(r,\mu)$:

$$\operatorname{ch} U(r,\mu) = \sum F_{\mathrm{Des}(w)}.$$

where F_D denotes a fundamental quasi-symmetric function, the sum runs over all vacillating tableaux w of length r and shape μ and $\mathrm{Des}(w)$ denotes the descent set of w.

Among others, this property justifies our bijection to be called "Sundaram-like", as she described a similar bijection for the defining representation of the symplectic group in her thesis [13]. There exists a similar (but less complicated) definition for descents in oscillating tableaux, which are used in the symplectic case instead of vacillating tableaux, and which Sundaram's bijection preserves. Thus there also exists a similar quasi-symmetric expansion of the Frobenius character, obtained by Rubey, Sagan and Westbury in [10].

In dimension 3 vacillating tableaux are Riordan paths: lattice paths with north-east, east and south-east steps, no steps below the x-axis and no east steps on the x-axis. This special combinatorial structure led to stronger results there, as we can use it to give an essentially different description of our Bijection B that proves further properties.

Vacillating tableaux of shape $\emptyset$ and dimension 3 can be regarded as paths that end at the x-axis, and can thus be concatenated naively. We define concatenation on standard Young tableaux by writing them side by side and adjusting entries to obtain the following: concatenation of vacillating tableaux corresponds to concatenation of standard Young tableaux (see Theorem 4.3.2). A similar property holds for Sundaram's bijection.

Ignoring the orthogonal Littlewood-Richardson tableaux, our bijection describes an insertion algorithm of vacillating tableaux to standard Young tableaux describing the decomposition of $U(r,\mu)$ (see Corollary 4.3.6). So does Sundaram's bijection for the symplectic group, mapping oscillating tableaux to standard Young tableaux. Due to the exceptional isomorphism $\mathfrak{sl}_2 \cong \mathfrak{so}_3$, our algorithm can also be regarded as an insertion algorithm for the decomposition of the isotypic components in tensor powers of the adjoint representation of SL_2. Thus, our algorithm is a first step towards a resolution of a closely related problem of Stembridge. He shows in [12, Theorem 6.2] that such an algorithm must also exist for the adjoint representation of GL_n, but so far, no explicit description has been found.

In the first Chapter we give background information and motivation.

In Chapter 2 we consider orthogonal Littlewood-Richardson tableaux. We define Kwon's tableaux, give an explicit description of them and present our new essentially different tableaux. To prove that they are indeed in bijection with Kwon's tableaux we provide Bijection A.

In Chapter 3 we formulate our main bijection, Bijection B, described by Algorithm 3.

In Chapter 4 we consider only dimension 3, provide an alternative Algorithm describing Bijection B in dimension 3 and prove further properties that are only conjectured for higher dimensions.

Chapter 1

Background

In this chapter we give background information and motivation and define the objects we will use in the following chapters in order to clarify notation.

1.1 Schur-Weyl duality

Considering the general linear group we start with the "classical Schur-Weyl duality"

$$V^{\otimes r} \cong \bigoplus_{\substack{\lambda \vdash r \\ \ell(\lambda) \leq n}} V^{\mathrm{GL}}(\lambda) \otimes S(\lambda).$$

Here V is a complex vector space of dimension n. The general linear group $\mathrm{GL}(V)$ acts diagonally (and on each position by matrix multiplication) and the symmetric group $\mathfrak{S}_r$ permutes tensor positions. Thus we consider a $\mathrm{GL}(V) \times \mathfrak{S}_r$ representation. $V^{\mathrm{GL}}(\lambda)$ is an irreducible representation of $\mathrm{GL}(V)$ and $S(\lambda)$ is a Specht module.

Now we consider a vector space V of odd dimension $n = 2k+1$. To obtain a similar decomposition, we use the restriction from $\mathrm{GL}(V)$ to $\mathrm{SO}(V)$

$$V(\lambda) \downarrow^{\mathrm{GL}(V)}_{\mathrm{SO}(V)} \cong \bigoplus_{\substack{\mu \text{ a partition} \\ \ell(\mu) \leq k}} c^{\mu}_{\lambda}(\mathfrak{d}) V^{SO}(\mu), \tag{1.1}$$

where $\mathrm{c}^{\mu}_{\lambda}(\mathfrak{d})$ is the multiplicity of the irreducible representation $V^{\mathrm{SO}}(\mu)$ of $\mathrm{SO}(V)$ in $V^{\mathrm{GL}}(\lambda)$. For $\ell(\lambda) \leq k$ this simplifies to the classical branching rule due to Littlewood.

Combining Schur-Weyl duality and the branching rule stated above we obtain an isomorphism of $\mathrm{SO}(V) \times \mathfrak{S}_r$ representations

$$V^{\otimes r} \cong \bigoplus_{\substack{\lambda \vdash r \\ \ell(\lambda) \leq n}} \Big(\bigoplus_{\substack{\mu \text{ a partition} \\ \ell(\mu) \leq k}} c^{\mu}_{\lambda}(\mathfrak{d}) V^{SO}(\mu) \Big) \otimes S(\lambda) = \bigoplus_{\substack{\mu \text{ a partition} \\ \ell(\mu) \leq k}} V^{SO}(\mu) \otimes U(r,\mu)$$

with isotypic components of weight μ

$$U(r,\mu) = \bigoplus_{\substack{\lambda \vdash r \\ \ell(\lambda) \leq n}} \mathrm{c}^{\mu}_{\lambda}(\mathfrak{d}) S(\lambda).$$

The isomorphism of SO(V) representations (e.g. Okada [7, Cor. 3.6]),

$$V^{SO}(\mu) \otimes V \cong \bigoplus_{\substack{\ell(\lambda)\leq k\\ \lambda=\mu\pm\square\\ \text{or } \lambda=\mu \text{ and } \ell(\mu)=k}} V^{SO}(\lambda)$$

implies that a basis of $U(r,\mu)$ can be indexed by so called vacillating tableaux of shape μ, defined in Section 1.3. Kwon defined orthogonal Littlewood-Richardson tableaux, as set that is counted by $c^{\mu}_{\lambda}(\mathfrak{d})$. We present Kwon's definition, as well as a new combinatorial description in Section 2.1 and introduce or new alternative tableaux in Section 2.2. A basis of $S(\lambda)$ can be indexed by standard Young tableaux. Therefore we are interested in a bijection between vacillating tableaux and pairs that consist of a standard Young tableau and an orthogonal Littlewood-Richardson tableau.

Moreover we introduce descent sets for vacillating tableau (see Section 1.3). We show that our bijection preserves these descents, and follow the approach taken by Rubey, Sagan and Westbury [10] for the symplectic group. This enables us to describe the quasi-symmetric expansion of the Frobenius character (see the textbook by Stanley [11]). Recall that the Frobenius character can can be defined by the requirement that it be an isometry and

$$\operatorname{ch} S(\lambda) = s_\lambda = \sum_{Q\in \mathrm{SYT}(\lambda)} F_{\mathrm{Des}(Q)}$$

where s_λ is a Schur function, $\mathrm{Des}(Q)$ denotes the descent set of a standard Young tableau (see Section 1.3.1) and F_D is the *fundamental quasi-symmetric function*

$$F_D = \sum_{\substack{i_1\leq i_2\leq\cdots\leq i_r\\ j\in D\Rightarrow i_j<i_{j+1}}} x_{i_1}x_{i_2}\dots x_{i_r}.$$

Therefore we obtain the following theorem.

Theorem 1.1.1.

$$\operatorname{ch} U(r,\mu) = \sum F_{\mathrm{Des}(w)},$$

where the sum runs over all vacillating tableaux w of length r and shape μ and $\mathrm{Des}(w)$ *is the descent set of w.*

1.2 (Skew) (Semi)standard Young Tableaux

We now introduce some well known concepts in order to clarify notation. For a textbook treatment see [11].

Definition 1.2.1. A *partition* $\lambda \vdash n$ of a nonnegative integer n is a sequence of positive integers $(\lambda_1, \lambda_2, \dots, \lambda_k)$ such that $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k > 0$ and $\lambda_1 + \lambda_2 + \cdots + \lambda_k = n$. The length $\ell(\lambda)$ of a partition λ is the number of integers in this sequence namely k.

A *Young diagram* of a partition λ is a collection of left-adjusted cells such that each row consists of λ_i cells.

The conjugate partition λ' is the partition belonging to the transposed Young diagram of the partition λ.

Let μ and λ be partitions such that $\mu \subseteq \lambda$ (thus $\mu_i \leq \lambda_i$). The *skew shape* $\lambda \backslash \mu$ is the Young diagram of λ with the cells of the Young diagram of μ missing. The partition μ is the inner shape while the partition λ is the outer shape.

A *horizontal strip* is a skew shape such that no two cells are in the same column.

Definition 1.2.2. A *semistandard Young tableau* of shape λ is obtained by a filling of the cells (with natural numbers) of the Young diagram of shape λ such that each row is weakly increasing and each column is strictly increasing.

We also consider *skew semistandard tableaux* where we take the Young diagram of a skew shape instead. We sometimes regard the missing cells as empty cells.

A *reversed (skew) semistandard tableau* is a filling such that each row is weakly decreasing and each column is strictly decreasing.

The *type* of a (reversed) semistandard Young tableau is $\mu = (\mu_1, \mu_2 \dots \mu_l)$ where μ_i is the number of i's in the tableau.

A *standard Young tableau* of shape λ is a semistandard Young tableau with entries $1, 2, \dots, |\lambda|$. Thus rows are also strictly increasing. We write $\mathrm{SYT}(\lambda)$ for the set of standard Young tableaux of shape λ.

A tableau is *column* (respectively *row*) strict if its columns (respectively rows) are strictly increasing.

By abuse of notation we call a horizontal strip in a tableau a collection of entries whose cells form a horizontal strip in the Young diagram.

Definition 1.2.3. The *Robinson-Schensted correspondence* maps a word $w_1, \dots, w_m$ with $w_i \in \mathbb{N}$ to a pair (P, Q) consisting of a semistandard Young tableau P, the insertion tableau, and a standard Young tableau Q, the recording tableau. (If and only if w is a permutation the insertion tableau P is also a standard Young tableau.)

To construct it we start with empty tableaux P and Q. We insert positions w_i of w from left to right into P. We insert w_i into the first row using the following procedure:

Element e gets inserted into row j as follows:

- If all elements in row j are smaller than or equal to e, (or row j is empty) place e to the end of row j.
- Otherwise search for the leftmost element f, that is larger than e, in row j. Put e to its spot and insert f into row $j+1$ using the same procedure again. We say that f got "bumped" into the next row.

Insert i into Q, where a new cell in P was added.

Definition 1.2.4. The *reading word* of a (skew) (semi)standard Young tableau is the word obtained by concatenating the rows from bottom to top.

Definition 1.2.5. A word w with entries in the natural numbers $w_1, w_2, \dots, w_l$ is called a Yamanouchi word (or lattice permutation) if for all i and any initial sequence s the number of i's in s is at least as great as the number of $(i+1)$'s in s.

A word $w_1, w_2, \dots, w_m$ is a *reverse Yamanouchi word* if $w_m, \dots, w_2, w_1$ is Yamanouchi.

For reverse Yamanouchi words the following theorem holds (see [6]):

Theorem 1.2.6. *If and only if a word w is a reverse Yamanouchi word, the insertion tableau P obtained by Robinson-Schensted is of the form*

1	⋯		*1*
2	⋯	*2*	
3	·	*3*	
⋮			

.

1.2.1 Descents of Standard Young Tableaux

Definition 1.2.7. Let $Q \in \mathrm{SYT}(\lambda)$ be a standard Young tableau. An entry j is a *descent* if $j+1$ is in a row below j. We define the *descent set* of Q as: $\mathrm{Des}(Q) = \{j : j \text{ is a descent of } Q\}$.

Example 1.2.8. The following standard Young tableau has descent set $\{2, 3, 5, 7, 12\}$. Descents j are bold, $j+1$ are italic.

1	**2**	10	11	**12**	14
3	**5**				
4	**7**				
6	9				
8	*13*				

1.2.2 Concatenation of Standard Young tableaux

Definition 1.2.9. The *concatenation* Q of two standard Young tableaux Q_1 and Q_2 is obtained as follows. First add the largest entry of Q_1 to each entry of Q_2 to obtain the tableau $\widetilde{Q_2}$. Then append row i of $\widetilde{Q_2}$ to row i of Q_1 to obtain Q.

This procedure is associative, thus we can consider the concatenation of several standard Young tableaux. We say a standard Young tableau Q is the concatenation of m standard Young tableaux if we can find standard Young tableaux $Q_1, \ldots, Q_m$ such that Q is the concatenation of those. We will be interested only in those concatenations where all tableaux have either rows of even length, or row lengths of the same parity, each.

Example 1.2.10. We concatenate two standard Young tableaux

1	2	5	6	9	10
3	4				
7	8				

and

1
2
3
4
5

and obtain

1	2	5	6	9	10	11
3	4	12				
7	8	13				
14						
15						

.

The first tableau itself is a concatenation of standard Young tableaux. The parts are the tableau containing only numbers 1 up to 8 and two single cells containing 1. If one is interested in a concatenation of tableaux with row lengths of the same parity, we can also take the tableau containing numbers 1 up to 8 and as second tableau, the one rowed tableau containing 1 and 2. (Empty rows j are counted as rows of even length for $j < n$.)

1.3 Vacillating Tableaux

We define vacillating tableaux (as defined by Sundaram in [14, Def. 4.1]) in three different ways, once as sequence of Young diagrams, once in terms of highest weight words and once as certain k-tuples of lattice paths.

Definition 1.3.1. 1. A ($(2k+1)$-*orthogonal*) *vacillating tableau* of length r is a sequence of Young diagrams $\emptyset = \mu^0, \mu^1, \ldots, \mu^r = \mu$ each of at most k parts, such that:

- μ^i and μ^{i+1} differ in at most one cell,

- $\mu^i = \mu^{i+1}$ only occurs if the kth row of cells is non-empty.

The partition belonging to the final Young diagram μ is the *shape* of the tableau.

2. A ((2k+1)-*orthogonal*) *highest weight word* is a word w with letters in $\{\pm 1, \pm 2, \dots, \pm k, 0\}$ of length r such that for every initial segment s of w the following holds (we write $\#i$ for the number of i's in s):

   - $\#i - \#(-i) \geq 0$,
   - $\#i - \#(-i) \geq \#(i+1) - \#(-i-1)$,
   - if the last letter is 0 then $\#k - \#(-k) > 0$.

   The partition $(\#1 - \#(-1), \#2 - \#(-2), \dots, \#k - \#(-k))$ is the *weight* of a highest weight word. The vacillating tableau corresponding to a word w is the sequence of weights of the initial segments of w.

3. Riordan paths are Motzkin paths without horizontal steps on the x-axis. They consist of up (north-east) steps, down (south-east) steps, and horizontal (east) steps, such that there is no step beneath the x-axis and no horizontal step on the x-axis.

   A k-tuple of Riordan paths of length r is a vacillating tableau of length r if it meets the following conditions:

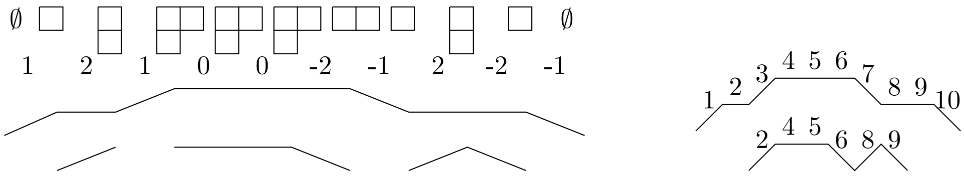
   - The first path is a Riordan path of length r.
   - Path i has steps where path $i-1$ has horizontal steps. Path i is never higher than path $i-1$.

   For a better readability we sometimes label the steps with $1, \dots, r$ in order to see which steps belong together and shift paths together.

   The corresponding highest weight word is described as follows: A value i is an up-step in path i and a horizontal step in paths 1 up to $i-1$. Similarly a value $-i$ is a down-step in path i and a horizontal step in paths 1 up to $i-1$ and a value 0 is a horizontal step in every path, including k.

By abuse of terminology we refer to all three objects as *vacillating tableaux*.

Example 1.3.2. The same object once written as a vacillating tableau, once as a highest weight word and once as a Riordan path. To the left we draw the Riodan path labeled and the second path shifted together in the way we described above.



1.3.1 Descents of Vacillating Tableaux

Definition 1.3.3. We define descents for vacillating tableaux using highest weight words. A letter w_i of w is a *descent* if there exists a directed path from w_i to w_{i+1} in the crystal graph for the defining representation of SO$(2k+1)$

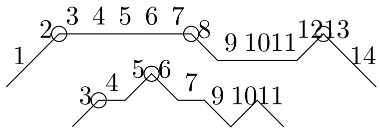
$$1 \to 2 \to \cdots \to k \to 0 \to -k \to \cdots \to -1$$

and $w_i w_{i+1} \neq j(-j)$ if for the initial segment $w_1, \ldots, w_{i-1}$ holds $\#j - \#(-j) = 0$.

We define the *descent set* of w as $\mathrm{Des}(w) = \{j : j \text{ is a descent of } w\}$.

In our tuple of paths a descent is a convex edge of consecutive steps, but not an up-step followed from a down-step on the bottom.

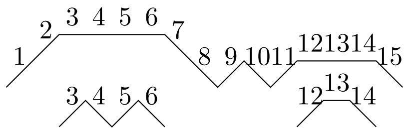
Example 1.3.4. The following vacillating tableau has descent set $\{2, 3, 5, 7, 12\}$. The corresponding positions are circled. Note that 10 is no descent, as 10, 11 are on bottom level. (It is no coincidence that the standard Young tableau in Example 1.2.8 has the same descents as they are assigned to each other by Bijection B.)



1.3.2 Concatenation of Vacillating Tableaux

Definition 1.3.5. The concatenation of vacillating tableaux of shape $\emptyset$ is obtained by writing them side by side. If we writing them labeled, we adjust the labels such that they are increasing from left to right.

Example 1.3.6. The following vacillating tableau is the concatenation of three vacillating tableaux, first the steps 1 to 8, then 9, 10, and third the steps 11 to 15. (We will see that it corresponds to the standard Young tableau of Example 1.2.10 under Bijection B.)



1.4 Crystal Graphs

In this section we summarize some properties of crystal graphs. In particular, we describe a certain crystal graph, that we need for defining orthogonal Littlewood-Richardson tableaux. For more information on crystals see the textbook by Hong and Kang [1].

Crystal graphs are certain acyclic directed graphs where vertices have finite in- and out-degree and each edge is labeled by a natural number. We only use crystal graphs whose vertices are labeled with certain tableaux.

For each vertex C there is at most one outgoing edge labeled with i. If such an edge exists we denote its target by $f_i(C)$. Otherwise $f_i(C)$ is defined to be the distinguished symbol $\varnothing$. Analogously there is at most one incoming edge labeled with i and we define $e_i(C)$ as the tableau obtained by following an incoming edge labeled with i. We denote by $\varphi_i(C)$ (respectively $\varepsilon_i(C)$) the number of times one can apply f_i (respectively e_i) to C.

We consider infinite crystal graphs. However, for the crystal graphs we consider, it holds that if we fix a natural number ℓ and delete all edges labeled with ℓ or larger, as well as all vertices that have incoming edges labeled with ℓ or larger, we obtain a finite crystal graph. Thus a lot of properties proven for finite crystal graphs hold also for our infinite crystal graphs.

The crystal graphs we consider are all tensor products of the following crystal graph.

Definition 1.4.1. The crystal graph of one-column tableaux is defined as follows:

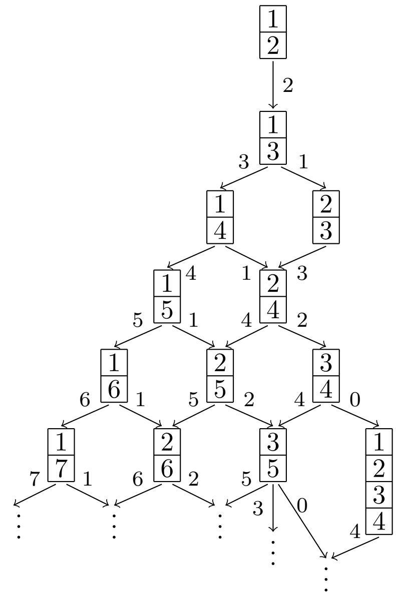


Figure 1.1: The component with even-length tableaux of a crystal consisting of one-column tableaux.

1. The vertices are column strict tableaux with a single column and positive integers as entries.
2. Suppose that $i \in \mathbb{N}, i > 0$ is an entry in a tableau C but $i+1$ is not. Then $f_i(C)$ is the tableau one obtains by replacing i by $i+1$. Otherwise $f_i(C) = \varnothing$.
3. Suppose that neither 1 nor 2 is an entry in a tableau C. Then $f_0(C)$ is the tableau one obtains by adding a domino $\begin{array}{|c|}\hline 1\\\hline 2\\\hline\end{array}$ on top of C. Otherwise $f_0(C) = \varnothing$.

See Figure 1.1 for an example.

We define now the tensor product of crystal graphs. We will use the tensor products of the crystal graph defined above for defining orthogonal Littlewood-Richardson tableaux.

Definition 1.4.2. The tensor product $B_1 \otimes B_2$ of two crystal graphs B_1 and B_2 is a crystal graph with vertex set $B_1 \times B_2$ and edges satisfying:

$$f_i(b \otimes b') = \begin{cases} b \otimes f_i(b') & \text{if } \varepsilon_i(b) < \varphi_i(b') \\ f_i(b) \otimes b' & \text{otherwise} \end{cases}$$
$$e_i(b \otimes b') = \begin{cases} e_i(b) \otimes b' & \text{if } \varepsilon_i(b) > \varphi_i(b') \\ b \otimes e_i(b') & \text{otherwise} \end{cases}$$

and

$$\varphi_i(b \otimes b') = \varphi_i(b) + \max(0, \varphi_i(b') - \varepsilon_i(b))$$
$$\varepsilon_i(b \otimes b') = \varepsilon_i(b') + \max(0, \varepsilon_i(b) - \varphi_i(b')).$$

Chapter 2

Alternative Orthogonal Littlewood-Richardson Tableaux

In this chapter we introduce a new kind of tableaux that count the multiplicities $c^{\mu}_{\lambda}(\mathfrak{d})$ in (1.1). To do so we first consider Kwon's orthogonal Littlewood-Richardson tableaux. Those are defined in a very general way using crystal graphs. As this definition is to abstract for our purpose, we find a new explicit formulation of them. This formulation we use to find a bijection (Bijection A) between Kwon's orthogonal Littlewood-Richardson tableaux, and our new alternative tableaux, which are defined terms of reversed skew semistandard tableaux with several conditions. This definition is similar to the one of Sundarams symplectic tableaux in [13], however the conditions are far more complicated.

The results from this chapter can be found in [2].

2.1 Kwon's orthogonal Littlewood-Richardson tableaux

In this section we will first present Kwon's orthogonal Littlewood-Richardson tableaux. This description is very general, so we give a new, explicit formulation of his orthogonal Littlewood-Richardson tableaux afterwards.

Although Kwon considers $\mathrm{O}(n)$, for odd n ($n = 2k+1$) we get $\mathrm{SO}(n)$ as a special case. In this case $V(\lambda) \downarrow^{\mathrm{O}(2k+1)}_{\mathrm{SO}(2k+1)}$ is an irreducible $\mathrm{SO}(2k+1)$ representation and every such representation is isomorphic to such a restriction (see for example Okada [7, Sect. 2.4]).

We start with introducing some notation we will use.

Definition 2.1.1. Let T be a two column skew semistandard tableau of shape $(2^b, 1^m)/(1^a)$, with $b \geq a \geq 0$ and $m > 0$.

The *tail* of T is the part where only the first column exists, that is, the lower m entries of the first column. The topmost tail position is the *tail root* and the rest of the tail is the *lower tail.*

The *fin* of T is the largest entry in the second column.

The *residuum* of T is the number of positions the second column can be shifted down while maintaining semistandardness. In particular, the residuum of T is at most $\min(a, m)$.

Definition 2.1.2. For a partition μ with at most k parts we define the crystal graph $B^{\mathfrak{d}}(\mu)$ as follows. It is the subgraph of the the tensor product of $n = 2k+1$ one column crystal graphs, whose vertices are tuples $(T_1, T_2, \ldots, T_{\ell(\mu)}, S)$ of skew semistandard tableaux such that:

- Each T_j has shape $(2^{b_j}, 1^{\mu_j})/(1^{a_j})$, with $b_j \geq a_j \geq 0$, b_j, a_j even and residuum at most one.
- S is of rectangular outer shape and has $n - 2\ell(\mu)$ (possibly empty) columns, all whose lengths have the same parity. We say S is even if its columns have even length, and S is odd otherwise.

Lemma 2.1.3 (Defining Lemma, Kwon)**.** *The crystal $T^{\mathfrak{d}}(\mu)$ is the subgraph of $B^{\mathfrak{d}}(\mu)$ whose vertices are in the same component as one of the following highest weight elements:*

- *T_j has its left column filled with $1, 2, \ldots, \mu_j$ and its right column empty.*
- *Either S is empty or S is a single row of $n - 2\ell(\mu)$ entries equal to 1.*

Definition 2.1.4. The set of *orthogonal Littlewood-Richardson tableaux* is

$$\mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d}) = \{L \in T^{\mathfrak{d}}(\mu) : \ i \text{ occurs in } L \text{ exactly } \lambda'_i \text{ times and } \varepsilon_i(L) = 0 \text{ for } i \neq 0\}$$

with $\ell(\lambda) \leq n = 2k + 1$ and $\ell(\mu) \leq k$.

As announced before the set of orthogonal Littlewood-Richardson tableaux is counted by $\mathrm{c}^{\mu}_{\lambda}(\mathfrak{d})$. See [5, Theorem 5.3]. This is one of the main results of [5].

Theorem 2.1.5 (Kwon). $\mathrm{c}^{\mu}_{\lambda}(\mathfrak{d}) = |\mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})|$

For two column skew shape semistandard tableaux we define admissibility which tells us if an element of $B^{\mathfrak{d}}(\mu)$ is in $T^{\mathfrak{d}}(\mu)$. To do so we need for a skew semistandard tableau consisting of a left and a right column $T = (T^L, T^R)$ the pairs $({}^LT, {}^RT)$ and (T^{L^*}, T^{R^*}):

Definition 2.1.6. Let $T = (T^L, T^R)$ be a two column skew semistandard tableau.

We define the pair $({}^LT, {}^RT)$ of two one-column, column strict tableaux as follows. Beginning at the bottom, we slide each cell of T^R down as far as possible, not beyond the bottom cell of T^L and so that the entry of its left neighbor is not larger. Then ${}^RT$ consists of all entries T^R, together with those in T^L that have no right neighbor. ${}^LT$ consists of the remaining entries in T^L.

If T has residuum 1, we define additionally the pair (T^{L^*}, T^{R^*}) of two one-column, column strict tableaux as follows. Beginning on the top, we slide each cell of T^L up as far as possible, not beyond the top cell of T^R and so that the entry of its right neighbor is not smaller. Then T^{L^*} consists of all entries in T^L, together with the largest entry in T^R that has no left neighbor. Note that such an entry must exist because T has residuum 1 and a is even, thus $a \geq 2$. T^{R^*} consists of the remaining entries in T^R.

See Figure 2.1 for examples.

Definition 2.1.7 (Kwon)**.** For a single column C, let $C(i)$ be the ith entry from the bottom and $\mathrm{ht}(C)$ its length.

Let T and U be two two-column skew semistandard tableaux with tails of length μ_T and μ_U such that $\mu_T \geq \mu_U > 0$ and residuum $r_T \leq 1$ and $r_U \leq 1$, respectively. The pair (T, U) is *admissible*, if the following conditions are met:

$$\text{(H)} \qquad \mathrm{ht}(T^R) \leq \mathrm{ht}(U^L) - \mu_U + 2r_T r_U$$

$$\text{(A1)} \qquad \begin{aligned} T^R(i) &\leq {}^LU(i) \quad \text{if } r_T \cdot r_U = 0 \\ T^{R^*}(i) &\leq {}^LU(i) \quad \text{if } r_T \cdot r_U = 1 \end{aligned}$$

(A2)
$$^{R}T(i+\mu_T-\mu_U) \leq U^L(i) \quad \text{if } r_T \cdot r_U = 0$$
$$^{R}T(i+\mu_T-\mu_U) \leq U^{L^*}(i) \quad \text{if } r_T \cdot r_U = 1$$

Let T be a two-column skew semistandard tableaux with tail of length $\mu_T > 0$ and residuum $r_T \leq 1$. Let S be a skew semistandard tableau of rectangular outer shape with first column S^L and columns with lengths of the same parity. The pair (T, S) is *admissible*, if the following conditions are met:

(H′)
$$\text{ht}(T^R) \leq \text{ht}(S^L) \quad \text{if } S \text{ is even}$$
$$\text{ht}(T^R) \leq \text{ht}(S^L) - 1 + 2r_T \quad \text{otherwise}$$

(A1′)
$$T^R(i) \leq S^L(i) \quad \text{if } S \text{ is even or } r_T = 0$$
$$T^{R^*}(i) \leq S^L(i) \quad \text{otherwise}$$

(A2′)
$$^{R}T(i+\mu_T-1) \leq S^L(i) \quad \text{if } S \text{ is odd and } r_T = 0$$
$$^{R}T(i+\mu_T) \leq S^L(i) \quad \text{otherwise}$$

Theorem 2.1.8 (Kwon)**.** *Let $L = (T_1, T_2, \ldots, T_{\ell(\mu)}, S)$ be a vertex in $B^{\mathfrak{d}}(\mu)$. Then L is a vertex of $T^{\mathfrak{d}}(\mu)$ if and only if any pair of successive tableaux in L is admissible.*

See Figure 2.1 for an example.

Remark 2.1.9. Let $L \in \text{LR}^{\mu}_{\lambda}(\mathfrak{d})$ be an orthogonal Littlewood-Richardson tableau. Moreover let $\tilde{L} = (T_j, T_{j+1}, \ldots, T_{\ell(\mu)}, S)$ be the tableau, which is obtained from L by deleting the first $j-1$ semistandard tableaux. Due to Theorem 2.1.8 $\tilde{L}$ is an orthogonal Littlewood-Richardson tableau in $\text{LR}^{\tilde{\mu}}_{\lambda}(\mathfrak{d})$, where $\tilde{\mu} = (\mu_j, \mu_{j+1}, \ldots \mu_{\ell(\mu)})$.

We give now an explicit description of Kwon's orthogonal Littlewood-Richardson tableaux. For it we need the concept of gaps and slots.

Definition 2.1.10. Let T be a semistandard tableau. A position $j > 1$ of T is a *gap* if $j-1$ is not in the same column as j. A position $j > 0$ of T is a *slot* if $j+1$ is not in the same column as j.

Note that above a gap there is either a slot or nothing and below a slot there is either a gap or nothing. In the first tableau of Figure 2.1 the 3 and the 8 in the first column and the 3 in the second column are slots, while the 7 in the first column is a gap and the 6 in the second column is both, a gap and a slot.

Theorem 2.1.11. *Let $\lambda \vdash r$, $\ell(\lambda) \leq n(= 2k+1)$, $\ell(\mu) \leq k$. Let $L = (T_1, T_2, \ldots, T_{\ell(\mu)}, S)$ be a vertex in $B^{\mathfrak{d}}(\mu)$. Then L is an orthogonal Littlewood-Richardson tableau in $\text{LR}^{\mu}_{\lambda}(\mathfrak{d})$ for $\text{SO}(n)$ if and only if for all i there are λ'_i i's in L and the following conditions are met:*

(H) $b_i \leq b_{i+1} - a_{i+1} + 2r_i r_{i+1}$ for $1 \leq i \leq \ell(\mu) - 1$.

(H′) $b_{\ell(\mu)} \leq ht(S^L)$ if S is even and $b_{\ell(\mu)} \leq ht(S^L) - 1 + 2r_{\ell(\mu)}$ if S is odd.

(S) S contains no gap.

(T1) Tableaux $T_1, T_2, \ldots, T_{\ell(\mu)}$ are of one of the following three types.

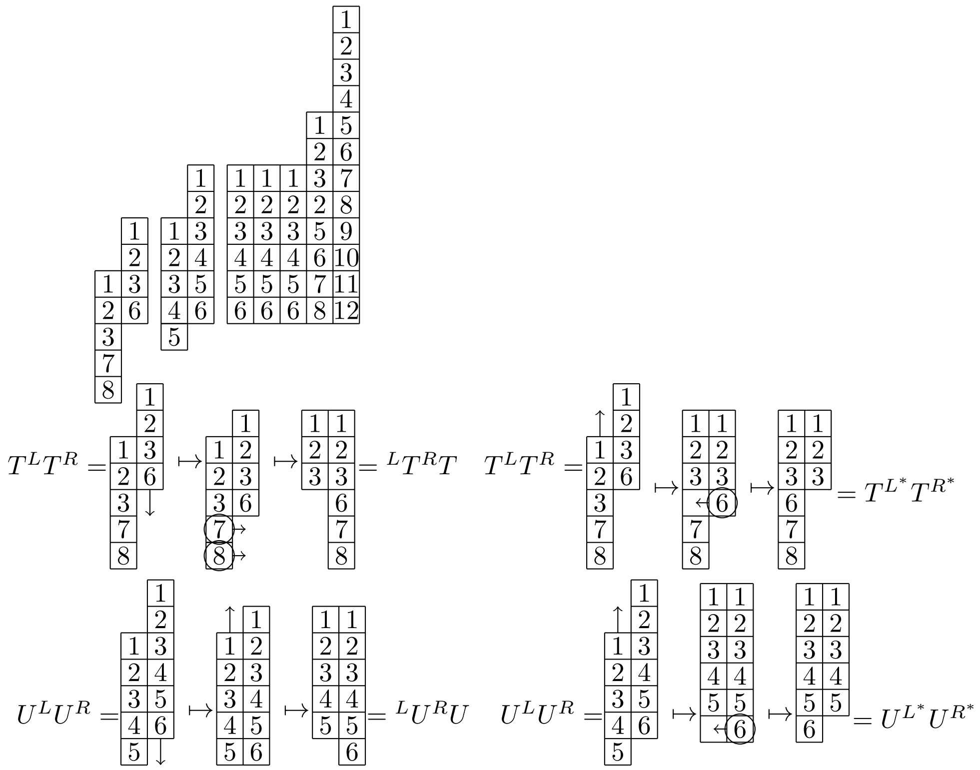


Figure 2.1: An orthogonal Littlewood-Richardson tableau in $\mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})$ with $n = 5$, $k = 3$, $\lambda = (12, 8, 8, 6, 6, 6, 6, 3, 3)$ and $\mu = (3, 1)$. We calculated the columns we need to prove its admissibility.

(a) Type 1 tableaux have residuum 0. Gaps can be only in the tail.

(b) Type 2 tableaux have residuum 1. Gaps can be only in the lower tail.

(c) Type 3 tableaux have residuum 1. The fin is a gap. Other gaps can be only in the lower tail.

If T_i is of type 3, $i < \ell(\mu)$, T_{i+1} has residuum 1 and the fin of T_i is not larger than the fin of T_{i+1}. If $T_{\ell(\mu)}$ is of type 3, S is odd.

If $T_{\ell(\mu)}$ is of type 1 and S is odd, the tail root is smaller than or equal to $S^L(1)$, the bottommost position in the first column of S.

(T2) The tails shifted together such that they share the top line form a semistandard Young tableau.

(G) For each gap j there is a slot $j-1$ in a column to the right. This can be in the same tableau T_i or in another one that is right of T_i in L including S. More precisely, if there are m gaps j there are m slots $j-1$ such that we can build pairs of a gap and a slot such that each slot is to the right of its gap.

Remark 2.1.12. Properties (H) and (H), as well as Properties (H′) and (H′) are just reformulations of each other. That is why we named them identically.

Lemma 2.1.13. *Let L be in $B^{\mathfrak{d}}(\mu)$. Then $\varepsilon_i(L) = 0$ for $i \neq 0$ if and only if* (S) *and* (G)*.*

Proof. If and only if $\varepsilon_j(C) > 0$ a column C contains a gap j. In this case $\varepsilon_j(C) = 1$. On the other hand if and only if $\varphi_j(C) > 0$ a column C contains a slot j. In this case $\varphi_j(C) = 1$.

The tensor product tells us $\varepsilon_j(b \otimes b') = \varepsilon_j(b') + \max(0, (\varepsilon_j(b) - \varphi_j(b')))$ and therefore $\varepsilon_j(b \otimes b') \geq \varepsilon_j(b')$.

For a a tensor product consisting of several columns to have $\varepsilon_j = 0$ this means that the first column needs to contain no gap and (G). Because S is a skew semistandard tableau and the rightmost column has no gaps, it cannot have gaps, because slots to the right are to big. □

Remark 2.1.14. This also shows that the filling of such a tableau is a partition.

Lemma 2.1.15. *Let $L = (T_1, T_2, \ldots, T_{\ell(\mu)}, S)$ be a tableau in $B^{\mathfrak{d}}(\mu)$ such that* (H), (H′), (S) *and* (G) *hold. Then if and only if* (A1) *and* (A1′) *hold also* (T1) *without the tail root condition for residuum zero tableaux holds.*

Proof. We first show inductively that the following two statements hold if and only if (A1) and (A1′) hold.

- Suppose T_i has residuum 1.
  - Then $T_i^{R^*}$ is T_i^R without the fin and ${}^L T_i$ is T_i^L without the lower tail. The tail root is not a gap.
  - There is no slot smaller than the bottommost position of $T_i^{R^*}$ in T_i^R or to the right. There is no slot smaller than the tail root in T_i^L or to the right.
  - If the fin is a gap, then T_{i+1} has also residuum 1, or if $i = \ell(\mu)$, S is odd.
- Suppose T_i has residuum 0.
  - Then ${}^L T_i$ is T_i^L without the tail.

– There is no slot smaller than the fin in T_i^R or to the right. There is no slot smaller than the bottommost position of ${}^LT_i$ in T_i^L or to the right.

This implies that there are no gaps at and above the positions in question, because slots to the right are to big.

In the base case $L = S$ we can argue that this is equivalent to S being a skew semistandard tableau.

In the induction step we consider T_1. (Compare with Remark 2.1.9.) If T_1 has residuum 1, it holds that:

- $T_1^{R^*}$ contains one position less than T_1^R. Let us call this position l_1. Suppose that l_1 is not the fin. In this case there exists a position l_3 directly below l_1. As l_1 is not in $T_1^{R^*}$, there exists a position l_2 in T_1^L, that is shifted next to l_3 when determining $T_1^{R^*}$. Therefore $l_1 < l_2 \leq l_3$. If $l_2 - 1$ is in T_1^L, it is at most one position above l_2, thus directly besides l_1, which is a contradiction. Therefore l_2 is a gap. If $l_2 - 1 = l_1$, either l_3 is a gap or l_1 is no slot. Thus either l_2 or l_3 is a gap with no slot in T_1. However l_3 is in $T_1^{R^*}$ and therefore smaller than or equal to the bottommost position of ${}^LT_2$ (or S^L if $\ell(\mu) = 1$, respectively). We have seen by induction that there are no smaller slots to the right. This is a contradiction.
- The bottommost position of $T_1^{R^*}$ (the position above the fin) is smaller than or equal to the bottommost position of ${}^LT_2$ (or S^L if $\ell(\mu) = 1$, respectively). Thus there is no slot that is small enough for this position or one above to be a gap.
- Because $T_1^{R^*}$ is T_1^R without the fin, the tail root is shifted above the fin when calculating $T_1^{R^*}$. Therefore it is smaller than or equal to the bottommost position of $T_1^{R^*}$. By the same argumentation as above, neither it nor a position above is a gap and no slot is smaller than it.
- If we consider the procedure to obtain ${}^LT_1$ we see that the fin is placed besides the tail root due to residuum 1, and therefore only the lower tail is shifted right.

If T_1 has residuum 0, it holds that:.

- The fin is smaller than or equal to the smallest slot to the right. Therefore it is no gap and there are no gaps above. The same holds for the position above the tail root.
- Due to residuum 0, nothing is shifted besides the tail root when calculating ${}^LT_1$, thus the whole tail changes column.

On the other hand, if those statements hold, the inequalities that hold for the bottommost positions of the considered columns, the column strictness and the lack of gaps imply (A1) and (A1′).

We prove now that those statements hold if and only if (T1) without the tail root condition for residuum zero tableaux holds. The statements about the slots imply where gaps are. On the other hand, if the gaps are where they are described in (T1) and (H) and (H′) hold, then we also get the inequalities between the slots in question. Finally the statements about ${}^LT_i$ and T_i^{R*} follow from the residuum and the places where a gap can be. □

Lemma 2.1.16. *Let $L = (T_1, T_2, \ldots, T_{\ell(\mu)}, S)$ be a tableau in $B^{\mathfrak{o}}(\mu)$ such that* (H), (H′), (S), (G) *and* (T1) *without the tail root condition for residuum zero tableaux hold. Then if and only if* (A2) *and* (A2′) *hold also* (T2) *and the tail root condition for residuum zero tableaux of* (T1) *hold.*

Proof. Due to what we have seen before about ${}^{L}T_i$ and $T_i^{R^*}$ this holds once we argue, that for residuum 1 tableaux the tail root is smaller than or equal to the fin.

The tail root condition for residuum zero tableaux and S odd is equivalent to the second condition of (A2′). □

Now Theorem 2.1.11 follows directly from Lemmas 2.1.13, 2.1.15 and 2.1.16.

We finish this section by proving further properties about orthogonal Littlewood-Richardson tableaux we will use later on.

Proposition 2.1.17. *If T_i is of type 2 or 3 the tail root is a slot.*

Proof. We have seen in the proof of Lemma 2.1.15, that the tail root is strictly smaller than the fin. Since the residuum is exactly 1, the entry below the tail root, if it exists, is larger than the fin. □

Proposition 2.1.18. *If the fin of a tableau T_i exists, it is even and not larger than the fin of T_{i+1}, which then also exists.*

Proof. The fin of T_i is even for type 1 or 2, as T_i^R has no gap and even length. We show for these cases that the fin is smaller than or equal to the fin of T_{i+1}. If T_i or T_{i+1} is of type 1, T_{i+1}^L without tail is at least as long as T_i^R by (H). Therefore, as $a_{i+1} \geq 0$, also T_{i+1}^R is at least as long as T_i^R. If both tableaux have residuum 1, T_{i+1}^L without tail plus 2 is at least as long as T_i^R by (H) and T_{i+1}^R is longer than T_{i+1}^L by at least 2. The claim follows as the fin of T_i is equal to the length of T_i^R and the fin of T_{i+1} is larger than or equal to the length of T_{i+1}^R.

If T_i is of type 3, we know that the fin of T_i is not larger than the fin of T_{i+1} by assumption. We show for this case that the fin is even. We do so by showing that any possible slot is odd.

Let T_j be the next tableau of residuum 0 to the right of T_i, if this exists, or $T_{\ell(\mu)}$, otherwise. Tableaux between T_i and T_j are therefore of type 3 or 2. Tableaux of type 3 have at least two odd slots, namely the position above the fin and the tail root. Tableaux of type 2 have at least one odd slot, namely the tail root. Other slots need to be at least as large as the fin. Therefore slots between T_j and T_i, that are small enough for the fin of one of those tableaux or T_i to be their slot, are also odd.

It remains to show, that there is no even slot right of T_j (and in T_j if it is of type 1), that is small enough for any fin of T_i or a tableau between T_i and T_j to be its gap.

If T_j is of type 2 or 3, it is directly left of S. As S contains no gap by (S), slots in S are in the bottom line. If S is odd, the slots of S are also odd. If S is even, T_j is of type 2, due to (T1) and any slot of S is larger than the fin of T_2 due to (H′).

If T_j is of type 1, slots of T_j are at least as large as the fin of T_{j-1} due to (H) and because the fin of T_{j-1} is not a gap, as it is of type 2 (T1). Due to (H) and (H′) (and because gaps are the fin or in the tail by (T1)) this also holds for slots further to the right. □

2.2 Our Alternative tableaux

In this section we define an alternative set of Littlewood-Richardson tableaux in terms of skew tableaux.

Definition 2.2.1. We define the set of alternative orthogonal Littlewood-Richardson tableaux aLR_λ^μ as follows. A tableau $L \in \mathrm{aLR}_\lambda^\mu$ is a reverse skew semistandard tableau of inner shape λ and type μ (thus the filling consists of μ_j j's, for all j). The outer shape has $2k+1$ possibly empty rows, whose lengths have all the same parity. The following two properties are satisfied.

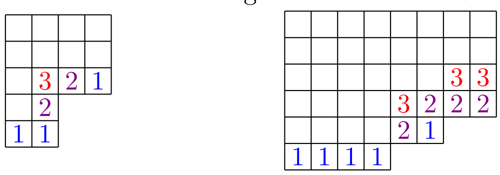
1. The reading word is a Yamanouchi word. This is satisfied if and only if the jth cell from left, labeled i is above the jth cell from left labeled $i-1$ for all $i > 1$.

2. We go through the reading word of L from right to left. Let p be the current position. We define a sequence v_p of positions of the reading word. The first entry of v_p is p. If $m-1$ entries of v_p are defined, let e be entry number $m-1$. We search now for entry number m. For that we consider entries whose letter is larger than the letter of e and which are in exactly $m-1$ sequences of positions right of p (thus sequences already defined). If this set is nonempty we search for the smallest letter in it and take the rightmost position with this letter as entry m. If it is empty, v_p has no more entries.

   Let r_p be the row p is in. Now we define the value o_p to be the number of entries in v_p with the following properties. It is the rightmost occurrence of its letter and if number m in v_p all $v_{\tilde{p}}$ with $\tilde{p} \neq p$ in the same row as p, have at most $m-1$ entries.

   We require $r_p \geq 2|v_p| - o_p$.

Example 2.2.2. Two alternative orthogonal Littlewood-Richardson tableaux.



We write the reading word as a sequence of entries l_p where l is the letter and p counts the position. The reading words are: $(1_1, 1_2, 2_3, 3_4, 2_5, 1_6)$ and $(1_1, 1_2, 1_3, 1_4, 2_5, 1_6, 3_7, 2_8, 2_9, 2_{10}, 3_{11}, 3_{12})$

Then we have the following v's, where rows are separated by semicolons: (1_6), (2_5), (3_4); $(2_3, 3_4)$; $(1_2, 2_5, 3_4)$, $(1_1, 2_3)$ and (3_{12}), (3_{11}); $(2_{10}, 3_{12})$, $(2_9, 3_{11})$, (2_8), (3_7); $(1_6, 2_{10}, 3_{12})$, $(2_5, 3_7)$; $(1_4, 2_9, 3_{11})$, $(1_3, 2_8, 3_7)$, $(1_2, 2_5)$, (1_1).

1_6 in the second tableau is in row 5, which is fine as 3_{12} is counted by o.

Proposition 2.2.3. *We can obtain the sequences v_e by using Robinson-Schensted on the reversed reading word of L. In particular v_e can be defined as the set of elements that got bumped during the insertion process of e.*

Therefore, by Theorem 1.2.6, the first property is satisfied if and only if the tableau one obtains by Robinson-Schensted on the reversed reading word is of the form as described in 1.2.6. This is satisfied if and only if every element j is bumped exactly j times. In terms of our v_e's this means that every j is in exactly j v_e's.

Proof. We show inductively that a position gets bumped if and only if it is in the current v_e. Therefore elements in the j-row were in j v_e's before.

For the base case we consider the first element of an v_e. This is always the one we are inserting. Thus it ends up in the first row. On the other hand an element that ends up in the first row, does so only during the insertion process of itself, thus when it is the first element of an v_e.

Now if an element is in j different v_e's, by induction hypothesis it got bumped j times thus it is now in row j. Now if it is element number $j+1$ in a v_e, it is the rightmost one of the smallest letter that is larger than the letter of element number j. As elements of the same value get inserted into a row from left to right in Robinson-Schensted, this is the rightmost element in the reading word. The same observation leads to the other direction. □

2.3 Bijection A

In this section we define a bijection (Bijection A) between Kwon's orthogonal Littlewood-Richardson tableaux and our new tableaux. We will use our new set of tableaux in the main bijection (Bijection B) to map pairs consisting of a standard Young tableau and a Littlewood-Richardson tableau to a vacillating tableau.

2.3.1 Formulation of Bijection A

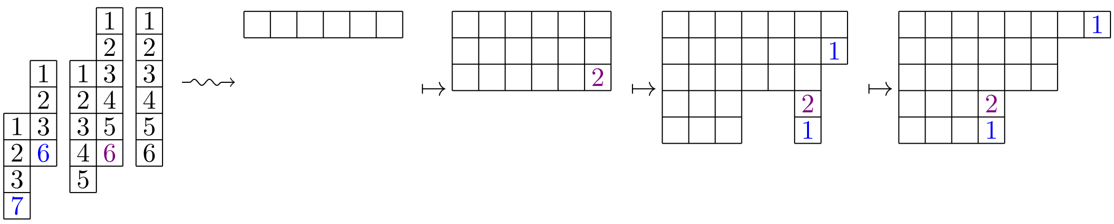
Bijection A is formulated by Algorithm 1. Its inverse is formulated by Algorithm 2. It maps an orthogonal Littlewood-Richardson tableau of Kwon in $\mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})$ to an alternative orthogonal Littlewood-Richardson tableau in $\mathrm{aLR}^{\mu}_{\lambda}$.

Algorithm 1: Orthogonal Littlewood-Richardson Tableaux: obtaining the alternative

input : orthogonal Littlewood-Richardson tableau $L = (T_1, T_2, \dots, T_{\ell(\mu)}, S) \in \mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})$
output: alternative orthogonal Littlewood-Richardson tableau $\tilde{L} \in \mathrm{aLR}^{\mu}_{\lambda}$
let $\tilde{L}$ be the Young diagram of S, reflected on $y = x$;
for $i = \ell(\mu), \ell(\mu) - 1, \dots, 1$ **do**
  for each l in T_i add an empty cell into column l of $\tilde{L}$;
  if *T_i has Type 1* **then** add below of cells coming from the tail of T_i a cell with entry i;
  else add below of cells coming from the lower tail and the fin of T_i a cell with entry i;
  sort each column such that empty cells are on top and entries are weakly decreasing;
  for *rows r from top to bottom* **do**
    while *there is a j left of an l such that $j < l$ in r* **do** `/* merge */`
      put the rightmost such l and the i from the same column one column to the left;
      shift cells that were below l upwards and sort the column to the left such that entries are weakly decreasing;
    while *in the row below of r are elements with no cell to the left* **do** `/* shift */`
      shift those elements and their lower neighbors one column to the left;
  while *not all rows have the parity of S* **do** `/* correct parity */`
    shift the rightmost i of the bottommost row with different parity as S to the next such row above;
return $\tilde{L}$;

2.3.2 Examples explaining Bijection A

Example 2.3.1. We consider an orthogonal Littlewood-Richardson tableau and apply Algorithm 1.



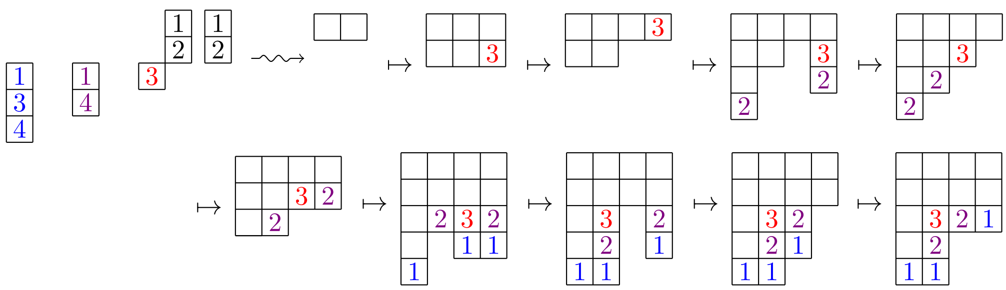
Doing so we insert first T_2 and then T_1. When inserting T_2, which is of type 2, we add a cell

Algorithm 2: Orthogonal Littlewood-Richardson Tableaux: obtaining the original

input : alternative orthogonal Littlewood-Richardson tableau $\tilde{L} \in \mathrm{aLR}^{\mu}_{\lambda}$
output: orthogonal Littlewood-Richardson tableau $L = (T_1, T_2, \dots, T_{\ell(\mu)}, S) \in \mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})$
for $i = 1, 2, \dots, \ell(\mu)$ **do**
  if *there is an odd number of i's in a row that is not row $2i+1$* **then**
    for *all such rows* **do** `/* correct parity */`
      put the rightmost such i one row below;
  for *rows r from bottom to top* **do**
    while *there are i's or vertically neighbored pairs i, j that can be shifted to the right such that there is still a cell directly above them and i or j is in r* **do**
      shift them one column to the right; `/* shift */`
    while *there are $i < j_1 < j_2$ in a column such that: the column to the right is shorter by at least two such that there exists a cell in the row above j_2, contains no i and no j_2; j_3, the position right of j_2, satisfies $j_2 > j_1 > j_3$ if it exists; j_2 is the topmost position in its column satisfying this and j_2 is in r* **do** `/* merge */`
      put i and j_2 one to the column to the right;
      shift cells below j_2 upwards and sort the column to the right such that entries are weakly increasing;
  for *i in $\tilde{L}$* **do** mark an unmarked empty cell in the same column, delete i and its cell;
  for *l; marked cells in column l* **do**
    insert a cell labeled l into the tail of the two rowed tableau T_i; delete it in $\tilde{L}$;
    shift the remaining cells upwards;
  if *row $(2i+1)$ or $(2i)$ are non-empty* **then**
    for each cell in column l in row $(2i+1)$ (respectively $(2i)$) insert a cell labeled l to the first (respectively second) column of T_i such that they are sorted increasingly;
  if *both new columns are of odd length (without tail)* **then** put the topmost tail position to the right column, shift the left column one position down;
reflect $\tilde{L}$ by $x = y$ and fill each column with $1, 2, \dots$ to obtain S;
let L be $(T_1, T_2, \dots, T_{\ell(\mu)}, S)$ and **return** *(L, Q)*

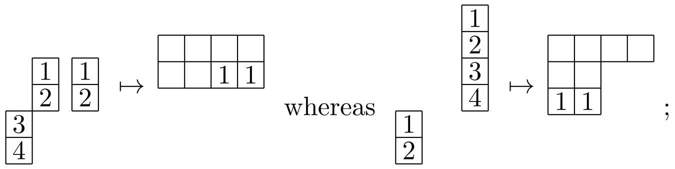
containing 2 below the cell coming from the fin and use neither *merge* nor *shift* nor *correct parity*. When inserting T_1, which is of type 3, we *shift* the pair $2, 1$ to the left and put the other 1 to a row above in *correct parity.*

Example 2.3.2. We consider another orthogonal Littlewood-Richardson tableau and apply again Algorithm 1.

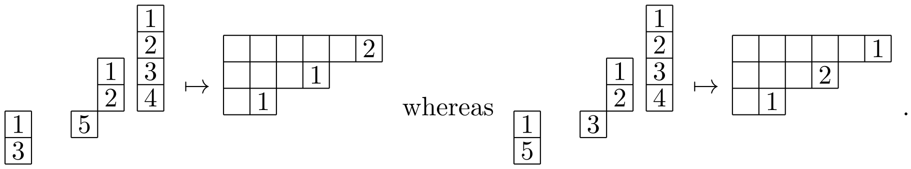


Doing so we insert first T_3 then T_2 and in the end T_1. All three of them are of type 1. When inserting T_3 we use only *correct parity* to put the 3 upwards. When inserting T_2 we first *shift* the pair 3, 2 to the left and then put the other 2 upwards in *correct parity*. When inserting T_1 we first *merge* the pair 3, 1 with the 2 to the left. Then we *shift* the pair 2, 1 to the left and in the end we put this 1 upwards in *correct parity*.

Example 2.3.3. The empty cells of our tableaux can be determined by the filling of Kwon's tableaux. However this shape does not define our tableaux by far. As the following tableaux show it neither defines where to add the filled cells:



nor does it define how to fill those cells:



2.3.3 Properties and Proofs for Bijection A

Theorem 2.3.4. *Algorithm 1 is well-defined and returns an alternative orthogonal Littlewood-Richardson tableau.*

Outline of the proof. We will prove this theorem by induction. In particular we show that after every iteration i of the outer for-loop the rules for alternative orthogonal Littlewood-Richardson tableaux are satisfied, if we would subtract $(i-1)$ from every entry. For the base case we get the shape of S reflected, which satisfies our conditions for $\mu = \emptyset$ and $k = 0$. The induction step is shown by the following lemmas. □

We state some properties following from the formulation of the algorithm first. We refer to parts or operations in the algorithms by the comments placed next to them.

Corollary 2.3.5. *1. In Algorithm 1 there are two types of rows that get longer during the inner for-loop. One type consists of those rows in which the new i's are inserted and the rows directly above. Those get longer by one for each such i. The other type consists of the bottommost two rows which get longer by values of the same parity.*

2. Correct parity *can be reformulated to the following and still leads to the same result.*

Go through $\tilde{L}$ from bottom to top. If the current row has a length of a different parity than S, put the rightmost i to the next row such that it is the leftmost i in this new row. (Shift other i's one column to the right.)

3. Unfilled positions form a Young diagram of a partition (and not a skew-shape).

Proof. 1. For cells that do not come from the lower tail and the tail root / fin we consider (H) and (H′). Due to those there is for each newly inserted empty cell an already inserted one coming from T_{i+1}. If $r_{T_i} = r_{T_{i+1}} = 1$ or S odd and $r_{T_{\ell(\mu)}} = 1$ this holds for the tail root and the non-tail-parts except for the fin. Otherwise this holds for the non-tail-parts.

Adding cells for the lower tail and the tail root / fin and adding cells with i corresponding to them extends columns by two. *Merge* or *shift* preserves this until the point where only an i is *shifted*. In this case this is the last movement of this i and it is still in the row below, the one that gets longer too.

2. Wrong parity is caused by columns getting longer by one. Therefore, if the row above the bottommost row with wrong parity has the right parity, there is also an i (an odd number of i's in fact). Iterating this argument completes the proof.

3. This follows directly from (G) and (S). □

Lemma 2.3.6. *Each step of the outer for-loop is well-defined if it adds a new T_i to an $\tilde{L}$ as demanded.*

Proof. There are three steps, in which this is not obvious:

- There is always an i in the current row when we *merge*.

  Columns that get longer by inserted cells of the non-tail parts cannot cause a *merge*-situation because then also the columns to the left get longer. Besides this, a column can only get longer by inserting an i, *merge* or *shift*. Only at insert i and *merge* a former inserted l can move to a different row. Thus only in those cases a *merge*-situation can arise. Therefore we can show inductively that there is always an i in such a column.

- There are always i's to find and places to put them at correct parity.

  For rows that are not the two bottommost ones this is follows from Corollary 2.3.5. We make rows longer in pairs. If we make them longer by an odd number, there needs to be enough space to put an element of the upper row to the lower one due to parity reasons. (If the latter got longer together with the one above, then this also got longer. We can iterate this argument.)

  Now we consider the bottommost rows. We start with the case that S is even and consider a newly inserted T_i. Suppose that the bottommost row is of odd length and contains no i. This means that an empty cell has been added but no i has been put below it. As non-tail parts of our T_i have even length, T_i needs to be of type 2 or 3.

  If T_{i+1} has residuum 0 (thus T_i is of type 2), the bottommost row so far had even length and was longer than or of equal length as T_i^R by (H). Therefore the fin of T_i is placed into the second row from bottom, with an i below. This i is put into the bottommost row, which is a contradiction. At this point there are no other j left or below to this.

  If T_{i+1} has residuum 1 it is also of type 2 or 3. In both cases the bottommost rows (those which consist of odd many empty cells, thus those above the fin of T_{i+1}) contain each one j increasingly from bottom to top. Now if we insert the new fin, it is inserted to one of these spots, as it is even and smaller than or equal to the fin of T_i. A sequence of *merge* puts the i into the bottommost row and the $i+1$ to the row above, as in each step a j is put into the row of an $j-1$.

  Now we consider the case that S is odd. We have to show that there are i's in a row, if this is of even length. Even columns without i cannot come from type 2 or 3 tableaux thus we have to consider type 1. If this is the first tableau from right thus directly left of S, the additional condition prevents this. Otherwise, we know that the tail root is smaller than or equal to the tail root to the right. Now we can argue exactly as above, but with the tail root instead of the fin.

- While-loops stop.

  The first while-loop stops after several steps because *merge* always works, as we have seen in the previous point, and moves i's to the left. As there are finitely many columns, this has to stop at some point.

  The second while-loop stops as after some steps everything is *shifted* to the left.

  The third while-loop stops as after some steps all rows have the same parity, because the parity of S equals the parity of the number of elements in $\tilde{L}$. This holds as the number of elements in S has the same parity as S and when inserting T_i we insert $2b_i - a_i + \mu_{T_i}$ empty cells and μ_{T_i} i's, which gives an even number of added cells to $\tilde{L}$. □

Lemma 2.3.7. *After each insertion of a T_i we obtain a reversed skew semistandard tableau.*

Proof. Each column is sorted such that fillings decrease and empty cells are on top as the columns get sorted after any operation that might change this. Each row is sorted the same way as due to Corollary 2.3.5 the empty cells build a Young diagram, thus are left in each row, and unsorted filled cells get eliminated by *merge*. Therefore it suffices to show that there is at most one j in each column.

As we insert at most one i to each column, there is at most one i in each column at the begin of an iteration. The same holds by induction for j with $j > i$.

We show that this holds also at the end of this iteration.

We note that the only situations where an l or i moves to another column are *merge*, *shift* and *correct parity*.

First we consider *merge*: Whenever there is a j in left of an l with $j < l$, this happens exactly if below the l is a newly inserted i or, if there was such a situation in the column to the right too, but not in the column of j, or if there was a shift of l and i. Therefore there cannot be an i in the column to the left because otherwise entries in this column would have moved one down too and $j < l$ would have caused a disorder before. Thus after *merge* the number of i in each column is still at most one. What remains to show is that there cannot be another l in the new column of l. Therefore we consider the position directly to the right of this other l. This needs to be smaller, as rows above are sorted and it needs to be larger as columns are sorted. This is a contradiction.

After *merge* the current row is sorted. Therefore *shifting* cells in the same row to the left does not increase the number of j's to two for any j and any column.

Finally we consider *correct parity*. Suppose an i is put into a column where already an i is. This would mean that there is not enough space for this i to be put into this row, if the other i would not be here. However we make rows longer in pairs, and if we make them longer by an odd number, there needs to be enough space to put an element of the upper row to the lower one due to parity reasons. □

Lemma 2.3.8. *After each insertion of a T_i the first property of alternative orthogonal Littlewood-Richardson tableaux holds.*

Proof. Due to (T2) each element i is inserted left below of the $(i+1)$ directly to the right in the tails. We show that operations in the outer for-loop do not change that.

If $(i+1)$ is still in the column where it was inserted or, due to *correct parity*, to the right, i gets inserted left of $(i+1)$. In this case neither *merge* nor *shift* can change this.

Now we consider the case that $(i+1)$ has changed column in *merge* or *shift*. If this happened and i is inserted to the right of it, we show that there needs to be an l above of i, thus *merge* also takes place for i. (If there is such an l, there was a position on top of the upper *merged* or *shifted* position, that now ends up to the right.) Where the empty cell belonging to i gets inserted, there needs to be an l or no cell and an empty cell to the left for the empty cells to form a tableau. No cell is not possible as $(i+1)$ was inserted to the same column (or to the right) and changed column in *merge* or *shift* and there is an empty cell to the left. A sequence of *merge* and *shift* will be followed by a sequence of *merge* by the same argument. (We always put two elements leftwards and the upper one will be the next candidate for *merge* as it is smaller than the element it is *shifted* to, because it was in the same column on top on it.)

What is left to consider is *correct parity.* In this case the row of i has odd (respectively even, depending on the parity of S) length and i is the rightmost position in it. Thus i is in an odd (respectively even) column. If $(i+1)$ is in a column to the right of i, i still ends up in the same column or to the left. If i is in the same column as $(i+1)$ we show that there cannot happen correct parity. The column where $(i+1)$ is now, got longer when it was inserted (or *merged/shifted to*). As it is still there and it is an odd (respectively even) column, the column to the left also got longer such that they had the same length. Thus it too contains an $(i+1)$, but no i, which is a contradiction.

l also changes column. If $(l-1)$ is in the same column, this was next to j before this column got longer through i. This means by induction $l-1 \leq j$ but since $j < l$ we obtain $l-1 = j$. Thus the $(l-1)$ in the original column of l is not its according $(l-1)$. (The $(l-1)$ in the column l is moved to is its according $(l-1)$.) ☐

Lemma 2.3.9. *After each insertion of a T_i also the second property of alternative orthogonal Littlewood-Richardson tableau holds.*

Proof. We start this proof by reformulating the second property:

Instead of putting elements into a v_e we can also mark them using the same rules. We remember how often an element was marked and count which element was marked at with position during considering e. Now we observe the following:

- It only matters how often an element is marked. It is not important by which elements it was marked.
- Whenever we consider an i and mark elements, if we mark an element the $(j+1)$-th time, we mark another, smaller element, the j-th time. Thus the number elements marked j times does never decrease.

Now we prove the statement.

First we have a closer look at what happens locally, when there is an i inserted (or *merge* happens) in one column but not in its neighbor.

To do so we first consider a column together with its left neighbor. We examine four elements in a pattern as below with $j_1 < j_2$, $j_3 < j_4$ and $j_2 \geq j_3$. *Merge* or insert i happens in the right column. Thus this is placed downwards by one. If $j_1 \geq j_3$ and $j_2 \geq j_4$ nothing changes, while if $j_1 < j_3$ and $j_2 \geq j_4$ or $j_2 < j_4$ we merge.

$$\begin{array}{|c|c|}\hline & j_4 \\ \hline j_2 & j_3 \\ \hline j_1 & \\ \hline\end{array}$$

is changed into: if $j_1 \geq j_3$ and $j_2 \geq j_4$: $\begin{array}{|c|c|}\hline j_2 & j_4 \\ \hline j_1 & j_3 \\ \hline\end{array}$ if $j_1 < j_3$ and $j_2 \geq j_4$: $\begin{array}{|c|c|}\hline j_2 & j_4 \\ \hline j_3 & \\ \hline j_1 & \\ \hline\end{array}$ if $j_2 < j_4$: $\begin{array}{|c|c|}\hline j_4 & j_3 \\ \hline j_2 & \\ \hline j_1 & \\ \hline\end{array}$

Now we determine which elements get marked if no other elements interfere:

- $j_1 \geq j_3$ and $j_2 \geq j_4$: $\{j_4\}$, $\{j_3, j_4\}$, $\{j_2\}$, $\{j_1, j_2\}$. After the insertion process this changes to $\{j_4\}$, $\{j_2\}$, $\{j_3, j_4\}$, $\{j_1, j_2\}$, which only changed the order.
- $j_1 < j_3$ and $j_2 \geq j_4$: $\{j_4\}$, $\{j_3, j_4\}$, $\{j_2\}$, $\{j_1, j_3, j_4\}$. After the insertion process this changes to $\{j_4\}$, $\{j_2\}$, $\{j_3, j_4\}$, $\{j_1, j_3, j_4\}$, which also only changed the order.

- For $j_2 < j_4$ we distinguish the cases $j_1 < j_3$ and $j_1 \geq j_3$: $\{j_4\}$, $\{j_3, j_4\}$, $\{j_2\}$, $\{j_1, j_3, j_4\}$ or $\{j_1, j_2, j_4\}$. After the insertion process this changes to $\{j_3\}$, $\{j_4\}$, $\{j_2, j_4\}$, $\{j_1, j_3, j_4\}$ or $\{j_1, j_2, j_4\}$, which is more than just a change of order but does not change anything about what is marked afterwards. j_3 and j_2 swap number of marked elements, which is allowed as j_2, which number increases, is one row below of where j_3 was. (The same row would have been sufficient.)

Now we consider a column together with its right neighbor. Again we examine four elements in a pattern as below with $j_1 < j_2$, $j_3 < j_4$, $j_1 \geq j_3$ and $j_2 \geq j_4$ (and therefore $j_2 > j_3$). *Merge* or insert i happens in the right column. Thus this is placed downwards by one. Everything ends up sorted, so no *merge* happens:

$$\begin{array}{|c|c|}\hline j_2 & j_4 \\ \hline j_1 & j_3 \\ \hline \end{array} \quad \text{is changed into} \quad \begin{array}{|c|c|}\cline{2-2} \multicolumn{1}{c|}{} & j_4 \\ \hline j_2 & j_3 \\ \hline j_1 & \multicolumn{1}{|c}{} \\ \cline{1-1} \end{array}\ .$$

Now we consider the marked elements in the insertion process: $\{j_4\}$, $\{j_2\}$, $\{j_3, j_4\}$, $\{j_1, j_2\}$ which changes to $\{j_4\}$, $\{j_3, j_4\}$, $\{j_2\}$, $\{j_1, j_2\}$ which is again only a change of the order.

As a second step we show that there are no relevant changes in those columns to the left and to the right. Once we have shown this, we can conclude, that $j > i$ still satisfy the third property, if we ignore elements counted by o.

To see this we can argue that anything even more to the left of a column that got changed is larger. Thus it marks even larger elements. In the case that it marks elements that would have been marked by j_1 to j_4 due to their change of rows, they simply swap which elements they mark. As they used to be in the same row, the third property still holds.

Everything more to the right of a column that got changed is smaller, and takes smaller elements, still it could be that the same kind of swapping occurs.

In the third step we have a closer look at what happens to i. If an i is inserted and not changed by *correct parity*, it is two rows below from where the elements one row above are. It can mark at most one element more, which still satisfies the third property.

Correct parity puts i one row up. If there are other i's in this row we can argue as above. Otherwise we see that it can mark at most one element more than those elements one row above. Lets call the rightmost one of them j. Suppose this j has only one row for every rightmost element in v_j and our i is not the rightmost i.

(If our i is the rightmost one, it can take only rightmost elements, and can take more elements as o. If j has more rightmost element i can take also more elements as o. This is because the element left of i can take at most as many elements as j but is the row of i now. Thus the "only" condition holds.)

For j to be taken in v_i, elements that are smaller or equal but more to the right have to be taken by other i. Let's call the leftmost such element m. Below m there is no row without a number because if there would be one, an element of v_j would be in the same row as one of v_m, that cannot satisfy the "only" condition. Thus when we insert another i below m that is not moved in correct parity above or besides m, either this i or another i left of it moves a position of v_j. Thus v_i has less elements too and satisfies the third condition.

Finally we consider elements that are counted by an o. Normally they just stay the same as every other element. When they are put one row down, it can happen that they count one time less as an o element, which is fine, as they also went down one row (and with them those which mark it). This happens if only this columns gains length and not the one directly to the left. □

Lemma 2.3.10. *$\tilde{L}$ has at most $2(k - i + 1) + 1$ rows, if S is odd, this number is met.*

Proof. Each column grows by adding empty cells and cells labeled i. *Merge* and *shift* only lead to grows of columns untouched so far. There are at most two numbers with the same value in

T_i, thus at most two new empty cells in each column. We show that if there are two, no i is inserted to the same column. Recall that i's are inserted below lower tail elements and the tail root or the fin, depending on the parity.

If a tableau has residuum zero, there cannot be an element that is in both, in the tail and in the right column. If a tableau has residuum 1, such a tail element could only be the tail root which produces no i either. The number of the fin cannot be in both columns, as the left column without the tail is shorter by at least two. Therefore no row can grow by more than two.

It remains to show the second claim. Thus we consider odd S and T_i that do not contain two 1's. Tableaux with residuum one contain a 1 in each column, as neither the position above the fin (which exists) nor the tail root nor any position above one of them is a gap. Thus we only consider tableaux of type 1.

If only the left column contains a 1, thus the right one is empty, we add i an empty cell in column 1.

Tableaux with no 1 in the left column consist only of the tail and a right column, if the tableau to the right had a left column larger than its tail (due to (H)). All those tail elements get inserted together with an i. Now the tail root needs to be smaller than or equal to all other tail roots by (T2). Residuum 1 tableaux produce two 1's, two 2's up to two such tail roots. Type 1 tableaux right of residuum 1 tableaux (respectively S) also do so due to (H) (respectively (H′)). Thus the first tableau of type 1, T_j, whose left column consists only of the tail inserts a j into row $2j+1$. Now as the second property of alternative tableau holds, i's that come afterwards will end up below, and their column will grow by two. (It is not possible that they end up there by a *shift* where only one element is *shifted*, as this would need two $(i+1)$'s belonging to one i which is a contradiction.) □

We have now proven every Lemma that proves Theorem 2.3.4. Next we show the same for the reversed algorithm.

Lemma 2.3.11. *Algorithm 2 is well-defined.*

More precisely, after each iteration i of the outer for-loop we obtain an alternative orthogonal Littlewood-Richardson tableau $\tilde{L}$ if we would decrease numbers by $i-1$.

Proof. Due to construction the algorithm is well-defined once we ensure that we have always enough cells to mark. We will see this during the proof. First we show that $\tilde{L}$ is a reverse skew semistandard tableau and the two properties of an alternative orthogonal Littlewood-Richardson tableau hold if they held before.

- Again, to show that $\tilde{L}$ is a reverse skew semistandard tableau, it suffices to show that everything is sorted and that there is at most one j in each column. Again no operation puts more than one i into the same column. Columns get sorted after deleting something, a violation of the row order is be prevented by *merge*, because a violation occurs exactly when the condition of *merge* arises.

  If an empty cell is erased left of another empty cell and thus shifts a cell labeled j_2 to an empty cell, or deleting shifts a smaller entry j_2 next to a bigger one j_3, this column is at least three cells larger, otherwise there would have been a *shift*. In this case we obtain *merge* and define j_1 to be the largest entry between j_2 and i.

- The only operation which can destroy the first property (that there is always a j below a $(j+1)$) is *merge* at j_2. This makes a problem if there is a (j_2+1) in the same column, and this j_2 is the one belonging to it. For (j_2+1) not to be taken instead of j_2 one of the following conditions must be met: either (j_2+1) is in the column to the right or $j_2 < j_4$ where j_4 is the position right of (j_2+1). In the former case j_2 belongs to that, which is a contradiction. In the latter case we obtain $(j_2 + 1) > j_4 > j_2$ which is also a contradiction as all three numbers are natural numbers.

- For the second property we can do a similar case study of local changes as we did before for Algorithm 1. However there are some steps we have to consider more precisely.

  When an i gets extracted, elements move one row up. This row however, was not necessary then, because the i causing this move needed two more rows for its formula, so the moved elements needed at least two less.

  For o we also argue similar as for Algorithm 1. If those which count for o are put upwards, but not the ones to the left, they are counted as o once more as before. This needs to be the case as we might not have this "not necessary" row in the o case.

Now we show that the row parity is constant. We shift one i for i's that are in odd sequences to the left. Thus the shifted i's shorten the row where they were by two (this i and their empty cell), while the other i's (an even number) shorten this row and the one above by an even number each.

The tableau has at most $2i + 1$ rows after an iteration as lower ones are taken as left and right column of the new tableau.

To see that there are enough cells to mark we consider the second condition of the alternative orthogonal Littlewood-Richardson tableau. This ensures one position to mark for positions, except for some o cases. In this cases *correct parity* puts it one row above. □

Lemma 2.3.12. *Each iteration of the outer for-loop produces a tableau of one of the three types.*

Proof. The shape follows from well-definedness and the last if-query once we can show that there are never one even and one odd row to be taken for the left and the right column of T_i. Thus we consider rows $2i + 1$ and $2i$ and distinguish two cases.

In the first case no i is put to row $2i + 1$ in correct parity. Therefore there are even many i's in row $2i$. Thus the parity of row $2i$ after extracting is the same as the parity of row $2i + 1$, as i's in row $2i + 1$ shorten both, row $2i$ and $2i + 1$. *Shift* or *merge* do not change this. In the second case an i is put from row $2i$ to row $2i + 1$ in correct parity. In the end this shortens row $2i$ by two, so parity is still preserved, by the same argument as in the first case.

Because elements that were put originally to the tail are larger than other elements, the residuum of T_i is 0 before the last if-query.

This also shows that a gap in the right column is the fin. If it is one the residuum is 1.

Moreover the tail root is not a gap for residuum one tableaux, as it comes from the left column without tail.

T_i is semistandard as row $2i + 1$ cannot be longer than row $2i$. □

Lemma 2.3.13. *The fin of such a tableau is even.*

Proof. The fin is even if it is no gap. If its a gap, rows $2i$ and $2i + 1$ are of odd length before extracting them. The leftmost i after *correct parity* is in an even column. (If S would be odd and this i would be in an odd column, it changes in correct parity. If S is even, it needs to be in row $2i + 1$ and was there before.) Neither *shift* nor *merge* can change this, as i's that are in the same row can be *shifted* to the right together and if *merge* occurs, there can either be another *merge* for the i to the left or they stay in the row where they are. As the parity of row lengths is constant and there are even many i's in other rows, this is sufficient. □

Lemma 2.3.14. *Let T_i be of type 3. If $i < \ell(\mu)$, T_{i+1} cannot be of type 1. If $i = \ell(\mu)$, S is odd.*

Proof. A tableau T_i of type 3 is formed in the last if-query where the tail root becomes the fin. The fin has to come from a row strictly above of row $2i$ as it is a gap. We show now that the bottommost row after extracting T_i, thus row $2i-1$ consist of odd many empty cells. As this will get into the left column of T_{i+1} we ensure residuum 1 or an odd S by what we have seen before.

As the fin of T_i is even, if something is extracted from row $2i-1$ during extracting T_i this row has odd many empty cells afterwards. Suppose that row $2i-1$ has even many empty cells and nothing is extracted from it. Row $2i+1$ has as many i's as there are $(i+1)$'s in row $2i$ and $(i+2)$'s in row $2i-1$. Otherwise i would not be *shifted* and there are to less j involved for *merge*. (If an i would not end up below an $(i+1)$, that is below an $(i+2)$, something would have been extracted from row $2i-1$.) Moreover we can conclude that no i was put to row $2i+1$ in *correct parity*. By the same argument i's that are in row $2i$ have exactly as many $(i+1)$'s in row $2i-1$. This number is even because nothing is changed in *correct parity*. We can conclude that row $2i$ contains odd many empty cells and several i's, while row $2i-1$ contains even many empty cells, the same number of $(i+1)$'s and an even number of i's. Thus the parity of the rows is different. This is a contradiction. □

Lemma 2.3.15. *If T_i is of type 1 and the tail root is a gap then either $i \neq \ell(\mu)$ or the tail root is odd.*

Remark 2.3.16. Once we have shown (H), (H′) and (G), it follows that the tail root needs an even slot to the right if it is a gap. In the case that S is odd, this would be the fin. Due to (H′) this is smaller by at least one than $S^L(1)$, which makes the tail root smaller than or equal to it.

Proof. We consider a tableau T_i of type 1 such that the tail root is a gap and even and $i = \ell(\mu)$. Thus i's are the last numbers in $\tilde{L}$. Therefore, at least one row ends with an odd position once T_i is extracted. Thus $\tilde{L}$ had rows with odd length. Now we consider row $2i+1$ and $2i$ before extracting T_i. Those get T^L without tail and T^R. Thus they are even. The leftmost i is in row $2i$ or above in order to take something from a different row, which is necessary for the gap. Thus row $2i+1$ has even length which is a contradiction. □

Lemma 2.3.17. (T2) *holds between two consecutive tableaux.*

Proof. All i's have corresponding $(i+1)$'s to the right. Thus, if this is not changed, they take smaller or equal numbers. We show that if *merge* occurs and i ends up in a column to the right, then either another *merge* occurs for $(i+1)$, or a *shift*, or that this was not the corresponding $(i+1)$.

Note that *correct parity* puts i's left so we do not need to consider it. Moreover i's below $(i+1)$'s are *shifted* together.

A *merge* situation in question occurs if $(i+1)$ is in the same column as i. It makes the column of $(i+1)$ shorter by two. $(i+1)$ could not be put one row above by *correct parity* as the row above has the same length. *Merge* puts $(i+1)$ one position upwards and puts i together with another j into the next column. If those columns have the same length after extracting i, there would be another $(i+1)$ belonging to another i due to the requirements of the positions right of j. Inductively this gives a contradiction. As those columns have different length such that all empty cells in the column of $(i+1)$ have right neighbors (because they had them before merging with i). $(i+1)$ either changes column by *shift* or by another *merge* when T_{i+1} is extracted.

Moreover we need to consider further applications of *merge* or *shift*. If i is *shifted* after *merge*, $(i+1)$ can follow this path. When it comes to another *merge* situation, the length of this column is not changed, thus $(i+1)$ can also introduce a *merge* situation if no $(i+1)$ is in this column (compare with above). □

Lemma 2.3.18. (H) *and* (H′) *hold between two consecutive tableaux.*

Proof. This follows as empty cells form a Young diagram of a partition. If row $2i+2$ was taken for T_i^L, row $2i+1$ is taken for T_{i+1}^R. The tail cannot consist of more than one position from this row, because to take something from this row an i must change row in *correct parity* or change column in the last if-query. The former cannot happen. The latter can happen only once. The only situation where T_i^R is shorter than T_{i+1}^L without tail is if in both tableaux there was something taken from their left column and put the right column in the last if-query. In this case both have residuum 1 and the column is longer by at most two, which is allowed. □

Due to construction also the following holds:

Corollary 2.3.19. *For each gap there is a slot to the right. Thus* (G) *holds. Moreover there are no gaps in* S*. Thus* (S) *holds.*

Therefore it follows that:

Theorem 2.3.20. *Algorithm 2 returns an orthogonal Littlewood-Richardson tableaux of shape* μ*.*

Theorem 2.3.21. *Algorithm 1 and Algorithm 2 are inverse.*

Proof. It suffices to show that one iteration of the outer for-loop (one insertion/extraction of a T_i) is inverse.

We insert an empty cell and one filled with i below and extract the same. Therefore what we have to show is that *merge*, *shift*, *correct parity* and dealing with the non tail parts are inverse. We can consider those separately as they do not interfere.

It is clear that the *shift*-procedures are inverse.

Now we consider the *merge* procedures. It is clear that they act inverse and that after *merge* in one algorithm we also *merge* in the other one. It remains to show that we do not *merge* in any other situation.

Merge in Algorithm 2 deals with an i. If it was not *merged* to get there, it was inserted there, or there was a *shift*. The former is not possible, as this would mean, that rows were not sorted before or the empty cells did not form a tableaux. The latter is prevented by $j_2 > j_1 > j_3$. *Merge* in Algorithm 1 happens if a row is not sorted. The only procedure that leaves a row unsorted in Algorithm 2 is *merge*.

To show see that *correct parity* is inverse we have to show that there cannot be an odd number of i's when there was no *correct parity* during inserting. (When we do correct parity in Algorithm 2, this changes the parity.) In Algorithm 1 rows $2i-1$ or above get longer if and only if they contain an i or the row below contains an i. Therefore only odd many i's produce a different parity in rows $1,\dots,2i-1$. Row $2i$ without i's has the same parity as row $2i+1$. If that is the wrong one, an i changes column. If row $2i$ has now an even number of i's in it, this is the wrong parity and another i changes column. Therefore only odd many i's produce a different parity and change place in *correct parity*.

For type 2 and type 3 tableaux we argue that inserting fin and lower tail is inverse to shift the fin to its place in the last if-query.

Columns (non tail parts) are placed to row $2i$ and $2i+1$ due to (H) and (H′). □

2.4 Results

With Theorems 2.3.4, 2.3.11, 2.3.20 and 2.3.21 we have proven the following Theorem. This is one of our main results.

Theorem 2.4.1. *Our alternative orthogonal Littlewood-Richardson tableaux* $\mathrm{aLR}^{\mu}_{\lambda}$ *are in bijection with Kwon's orthogonal Littlewood-Richardson tableaux* $\mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})$*. Therefore they also count the multiplicities* c^{μ}_{λ} *in* (1.1).

Chapter 3

A Bijection for $\mathrm{SO}(2k+1)$

In this chapter we use the alternative tableaux we have found in the last chapter to define our bijection in question. The formulation of alternative tableaux reduces our problem into finding a bijection between certain kinds of standard Young tableaux and vacillating tableaux with empty shape.

We first formulate the strategy of our bijection and give two examples. Then we concentrate on the core of our bijection, namely Bijection B. We formulate it in terms of Algorithm 3 and its reversed algorithm, Algorithm 4. We explain it using various examples and conclude the section about Bijection B by stating and proving properties of it. We continue proving further properties concerning a structure we call μ-horizontal strips, which turns out to be the link between Bijection A and Bijection B. Finally we give a short outlook and state some conjectures concerning additional properties, one of which we are able to prove for dimension 3 in the next chapter.

The results of this chapter can be mostly found in [2]. Some properties for dimension 3 can be found in [3].

3.1 Formulation of our bijection

Definition 3.1.1 (The Bijection for $\mathrm{SO}(2k+1)$)**.** We start with a pair (Q, L) consisting of a standard Young tableau Q in $\mathrm{SYT}(\lambda)$ and an orthogonal Littlewood-Richardson tableau L in $\mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})$.

First we use Bijection A (see Section 2.3) to change L into an alternative orthogonal Littlewood-Richardson tableau $\tilde{L}$. Now we use $\tilde{L}$ to obtain a larger standard Young tableau $\tilde{Q}$ with row lengths of the same parity as follows. If e is the largest entry in Q we add a cells labeled $e + (\mu_{j+1} + \cdots + \mu_{\ell(\mu)}) + 1$, $e + (\mu_{j+1} + \cdots + \mu_{\ell(\mu)}) + 2$, $\ldots$, $e + (\mu_{j+1} + \cdots + \mu_{\ell(\mu)}) + \mu_j$ to the spots where cells labeled j are in $\tilde{L}$, such that the numbers in the horizontal strip belonging to j are increasing from left to right. We obtain a new standard Young tableau $\tilde{Q}$ with the same shape as L. Moreover the μ largest entries form a μ-horizontal strip (see Section 3.3).

Now we distinguish two cases: If our resulting tableau $\tilde{Q}$ consists of even length rows this is the tableau we will use in Bijection B (see Section 3.2.1). Otherwise, thus when $\tilde{Q}$ consists of n odd length rows, we concatenate the one column tableau filled with $1, 2, \ldots, n$ from left to $\tilde{Q}$. We obtain an all even rowed standard Young tableau, which we will use in in Bijection B.

We continue applying Bijection B to $\tilde{Q}$ and obtain a vacillating tableau $\tilde{V}$ with shape $\emptyset$ and cut-away-shape μ (shape $\emptyset$ ending with $\mu_{\ell(\mu)}$ $(-\ell(\mu))$'s, $\ldots$, μ_2 (-2)'s and μ_1 (-1)'s, see

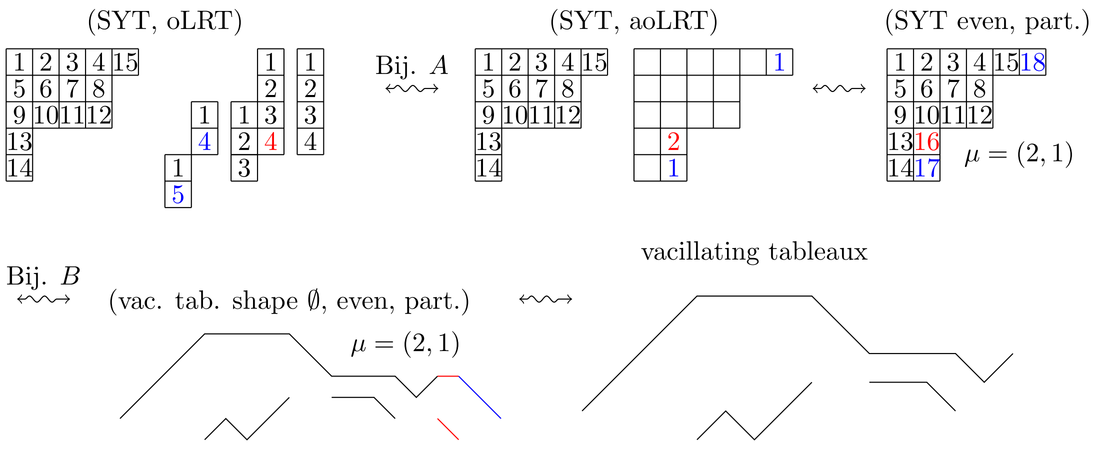


Figure 3.1: The strategy of our bijection outlined in an even case (with $r = 15$ and $k = 2$)

Section 3.3).

Once again we distinguish the two cases from before. If we did not concatenate with a column, we do not change $\tilde{V}$. If we concatenated a column to $\tilde{Q}$, we delete the first n entries of $\tilde{V}$. In this case those always are $1, 2, \ldots, k, 0, -k, \ldots, -2, -1$. Therefore we obtain once again a vacillating tableau $\tilde{V}$ with shape $\emptyset$ and cut-away-shape μ.

We finish our algorithm by deleting the last $|\mu| = \mu_1 + \mu_2 + \cdots + \mu_k$ entries to obtain a vacillating tableau V of shape μ and length $r = |\lambda|$.

In Figures 3.1 and 3.2 we illustrate this using an even example for $r = 15$, $k = 2$, $n = 2k+1 = 5$ and an odd example for $r = 17$, $k = 3$, $n = 2k + 1 = 7$. In Table 2 in the appendix we provide a list of all tableaux with $r = 3$ and $n = 5$.

As we know the inverse of Bijection A and Bijection B, the inverse bijection is easily defined:

Definition 3.1.2 (The other direction of our bijection)**.** We start with a vacillating tableau V of shape μ and length r, and add μ_k $(-k)$'s, μ_{k-1} $(-k + 1)$'s, ... and μ_1 (-1)'s to obtain a vacillating tableau V of shape $\emptyset$ and cut-away-shape μ. If this has odd length we furthermore add $1, 2, \ldots, k, 0, -k, \ldots, -2, -1$ in the front. Next we apply the inverse of Bijection B to obtain a standard Young tableau $\tilde{Q}$.

If we added $1, 2, \ldots, k, 0, -k, \ldots, -2, -1$ to $\tilde{V}$, we cancel the smallest n entries of $\tilde{Q}$ now. Those are in the first column in increasing order. If we did so, we furthermore reduce each entry of $\tilde{Q}$ by n afterwards, to obtain a standard Young tableau again.

We obtain Q by deleting the $|\mu|$ largest entries in $\tilde{Q}$. Q is a standard Young tableau of shape λ. Moreover we define $\tilde{L}$ to be the reverse skew semistandard tableau of the same outer shape as $\tilde{Q}$ and inner shape λ. We fill cells, where entries of μ_j are in $\tilde{Q}$ with j. Due to the properties of μ-horizontal strips, $\tilde{L}$ is an alternative orthogonal Littlewood-Richardson tableau.

Finally we apply the inverse of Bijection A to obtain L, an orthogonal Littlewood-Richardson tableau in $\mathrm{LR}^{\mu}_{\lambda}(\mathfrak{o})$ where λ is a partition such that $\lambda \vdash r$ and $\mu \leq \lambda$.

The strategy we use is the one we use in [2]. It is similar as in the case $n = 3$ in [3]. There are two main differences. The first is, that in [3] we do not calculate the alternative Littlewood-Richardson tableau but go directly to the μ-horizontal strip. The second is, that we attach numbers for odd tableaux to the left of Q in order to obtain an all even rowed tableau. However, in case $n = 3$ we know, that concatenating of standard Young tableaux where all row

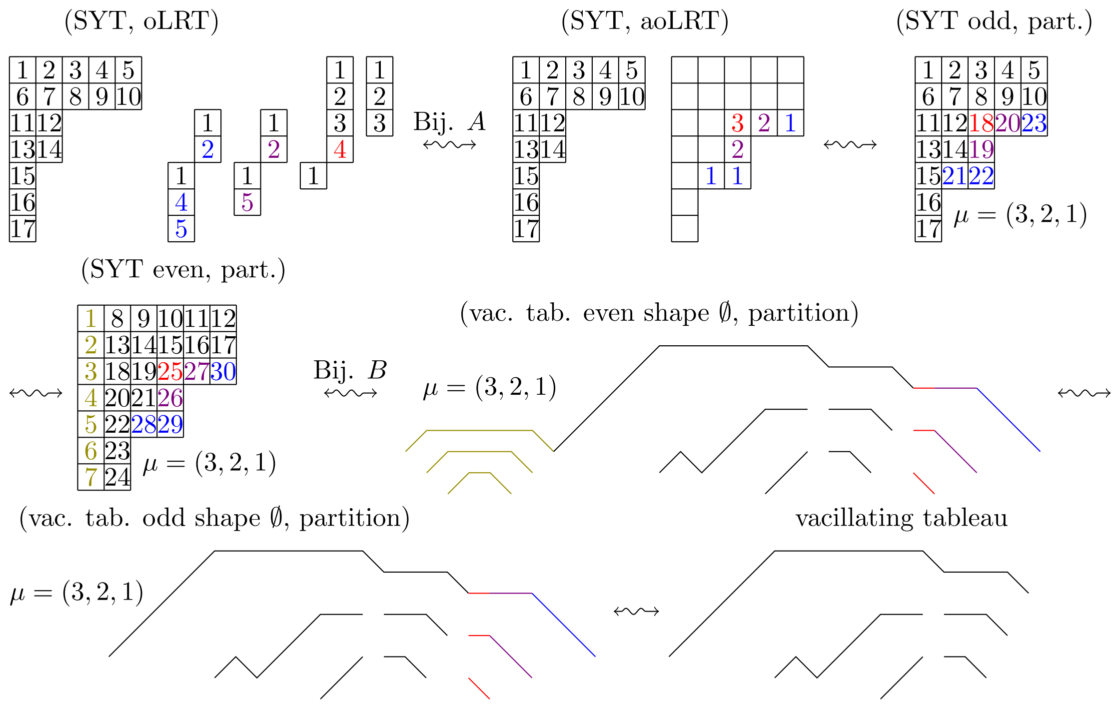


Figure 3.2: The strategy of our bijection outlined in an odd case (with $r = 17$ and $k = 3$)

lengths have the same parity corresponds to concatenation of vacillating tableaux with shape ∅. Therefore, for $n = 3$ both strategies are the same.

Theorem 3.1.3. *Let $\lambda \vdash r$, $\ell(\lambda) \leq n(= 2k+1)$, $\ell(\mu) \leq k$. The map defined in this section maps a pair (Q, L) consisting of a standard Young tableau Q in $\mathrm{SYT}(\lambda)$ and an orthogonal Littlewood-Richardson tableau L in $\mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})$ to a vacillating tableau of length r and shape μ. Moreover it is well-defined, bijective and descent preserving.*

Proof. We prove that every algorithm we use defines a well-defined mapping in Theorems 2.3.4, 2.3.11, 3.2.7 and 3.2.17. Those also show, together with Theorems 3.3.11 and 2.3.20, that they produce the desired objects.

To see that it is bijective, we argue that the algorithms we use are inverse in Theorems 2.3.21 and 3.2.22. Moreover the procedure we describe between alternative orthogonal Littlewood-Richardson tableaux and μ-horizontal strips is inverse by definition. The procedure we describe by adding and deleting the first positions is inverse by Theorem 3.2.24.

It is descent preserving as Bijection B is descent-preserving (see Theorem 3.2.23). □

3.2 Bijection B

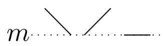
3.2.1 Formulation of Bijection B

Bijection B is formulated by Algorithms 3 and 4 which are inverse. It maps a standard Young tableau with $n = 2k + 1$ possibly empty rows, whose lengths are even, containing a μ-horizontal strip, to a vacillating tableau of dimension k, shape ∅ and cut-away-shape μ.

To formulate those algorithms we introduce some notation in Table 3.1. Note that at some points we have left to right opposites for those algorithms. When looking at weight $\emptyset$ words, which will not always be the case while executing Algorithms 3 and 4, the definitions are the same.

We refer to parts or operations in the algorithms by the comments placed next to them.

Table 3.1: Notation for Algorithms 3 and 4

A *labeled word* w with letters in $\{\pm 1, \dots, \pm k, 0\}$.	A word, where each letter is labeled by an integer $1 \leq i \leq r$ strictly increasing from left to right. Each position consists of a label and an entry. We denote by $w(p)$ the entry of w labeled with p.
A position q is on l-level m in Algorithm 3 (respectively Algorithm 4).	The maximum (respectively minimum) of the following two sums over entries with absolute value l is $-l \cdot m$ (respectively $+l \cdot m$). For the first sum we consider entries strictly to the *right* (respectively *left*) of q. For the second one we consider entries to the *right* (respectively *left*) including q. Illustration of positions on level m: 
A position q is a height violation in l.	The l-level of q is smaller than the $(l+1)$-level of q. If $w(q) = \pm(l+1)$ we take the $(l+1)$-level plus one instead.
Insert q with l.	We insert a new position with entry l and label q in that way, that the labels are still sorted.
Ignore q.	Act as if this position was not here, for example in level calculations.
A position p is a 3-row-position in j.	p is either the rightmost 0 of an odd sequence of 0's on j-level one or a 0 that is on j-level two or higher.
A position p is a 2-row-position in j.	p is either a j on j-level one or the leftmost 0 of a sequence of 0's.
A position p is in a j-even (respectively odd) position	The number of positions q strictly to the left with $w(q) \in \{0, \pm j\}$ is even (respectively odd).

3.2.2 Examples explaining Bijection B

We can draw labeled words like we draw vacillating tableaux as tuple of paths, compare with Example 1.3.2.

Example 3.2.1 (An easy example to motivate the Algorithm). We consider the following tableau for $n = 2k+1 = 7$, thus we are going to create $k = 3$ paths:

1	2	3	5	19	20
4	6	8	16	21	22
7	9	10	17		
11	12				
13	14				
15	18				

- We initialize the first path with up, down, up, down, ..., up, down-steps, labeled with the elements of the first row.

- We insert rows 2 up to $2k+1$ from top to bottom. For each row we insert pairs of two elements, starting with the rightmost pair, into the topmost path.

Algorithm 3: Standard Young Tableaux to Vacillating Tableaux

```
input  : n = 2k+1, standard Young tableau Q of at most n rows, all rows of even length
output: vacillating tableau V, dimension k, weight ∅, same number of entries as Q
let w be word (1,−1,...,1,−1) labeled by first row elements of Q;   /* insert row 1 */
for i = 2,3,...,n do                                                  /* insert row i */
  j := ⌊i/2⌋; unmark everything;
  if i even then change 0-entries of w into j,−j,...,j,−j;            /* initialize j */
  for pairs of elements a,b in row i, start with the rightmost, go to left do
    a_1 := a, b_1 := b, a_l := b_l := 0 for l = 2,3,...,j+1;
    if b is largest position so far then insert b_1 with −1;          /* b */
    let p be rightmost position so far, p̃ be next position left of p with w(p̃) ∈ {0,±j};
    while a_{j+1} < p or w(p) ∉ {0,±j} do
      if p < b_l, p ≠ a_l, w(p) = −l for an l < j, a_{l+1} = 0 then
        if p not marked, b_{l+1} = 0 then w(p) := −l−1, b_{l+1} := p;   /* b_{l+1} */
        else if p < a_l, p < b_{l+1} then w(p) := −l−1, a_{l+1} := p;   /* a_{l+1} */
      if i is even, w(p) ∈ {0,±j} then                                 /* i even */
        if b_j < p, w(p̃),w(p) = j,−j then              /* adjust separation point */
          for l < j change ±l on l-level 0 between p and p̃ into ±(l+1), if p < b_l,
           b_{l+1} = 0 ignore b_l, if p < a_l, a_{l+1} = 0 ignore a_l; mark changed
           positions; change −j,j between p and p̃ into 0,0;
        else if a_j < p, w(p̃),w(p) = j,−j then             /* mark it + connect */
          w(p̃),w(p) := 0,0; for l < j mark ±l on l-level 0 between p and p̃, if
           p < a_l, a_{l+1} = 0 ignore a_l;
        else if p = a_j, w(p̃) = 0 on j-level 1 then                 /* a_{j+1} 1 */
          w(p̃),w(p) := j,0, a_{j+1} := p̃;
        else if p < a_j, w(p) = −j, a_{j+1} = 0 then                  /* a_{j+1} 2 */
          w(p) := j, a_{j+1} := p;
        if p < b_j, b_{j+1} = 0 then b_{j+1} := p;                    /* b_{j+1} */
      if i is odd, w(p) ∈ {0,±j} then                                   /* i odd */
        if b_{j+1} < p, w(p),w(p̃) = 0,0, p j-even position on j-level 1 if b_j < p or 2 if
         p < b_j then                                   /* adjust separation point */
          for l < j change ±l on l-level 0 between p and p̃ into ±(l+1), if p < b_l,
           b_{l+1} = 0 ignore b_l, if p < a_l, a_{l+1} = 0 ignore a_l; mark changed positions;
        else if a_{j+1} < p < b_{j+1}, w(p) = j on j-level 1 for p < a_j or 0 for a_j < p
         then w(p̃),w(p) := 0,0;                                       /* connect */
        else if a_{j+1} < p < b_{j+1}, w(p̃),w(p) = 0,0, p j-even position on j-level 2 if
         p < a_j or 1 if a_j < p then                        /* mark it + separate */
          w(p̃),w(p) := −j,j; for l < j mark ±l on l-level 0 between p and p̃, if
           p < a_l, a_{l+1} = 0 ignore a_l;
        else if p < b_j, p ≠ a_j, w(p) = −j, a_{j+1} = 0 then
          if p not marked, b_{j+1} = 0 then w(p) := 0, b_{j+1} := p;    /* b_{j+1} */
          else if p < a_j, p < b_{j+1} then w(p) := 0, a_{j+1} := p;    /* a_{j+1} */
      if p = a_l on l-level 0, for an l < j, the l to the right is marked then
        mark a_l;                                                       /* mark a_l */
```

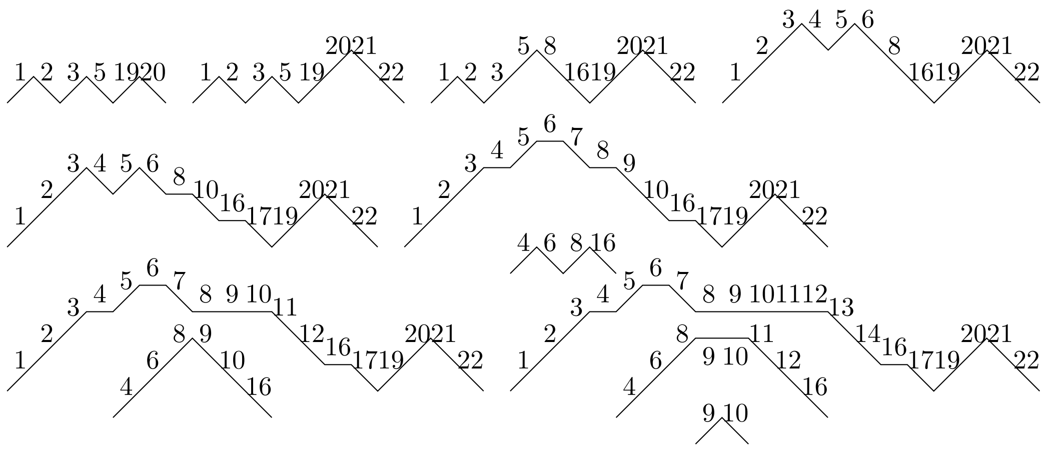
if *p height violation in l for an* $l < j$, *(*$p < a_l$ *or p not marked), if* $p < a_l$, $a_{l+1} = 0$ *ignore* a_l **then** `/* height violation */`
  $w(p) := l + 1$;
  if $a_{l+1} = 0$ **then** $b_{l+1} := 0$ **else** $a_{l+1} := 0$;
  if *i is even,* $a_{j+1} \neq 0$ **then** $w(a_{j+1}), w(p) := 0, 0$, $a_{j+1} := 0$;
  if *i is odd,* $w(\tilde{p}) = 0$ *on j-level* 0 **then** $w(\tilde{p}) := -j$, $b_{j+1} := 0$;
if *b is between p and the position to the left* **then** insert b_1 with -1; `/* b */`
else if *a is between those* **then** insert a_1 with -1; `/* a */`
let p be one position to the left in w, change $\tilde{p}$ according to it;
do one additional iteration of the inner for-loop with $a = b = 0$;
forget the labels of w, set $V = w$;
return V;

- When we insert a pair a, b into a path, we insert the new positions a and b with down-steps and
  - if we have not inserted into this path during the insertion process of the previous row, we change the down-step left of a pair (a, b), into an up-step.
  - otherwise we change the next down-step to the left of each, a and b into a horizontal step.

  If there is a path beneath, we insert these new horizontal-steps as a pair a, b into this path according to this rule.

- Whenever we finish inserting an odd row, we initialize a new path below (with up- and down steps) and label it with the horizontal steps of the bottommost path so far.

So the core of Algorithm 3 is an an insertion algorithm from standard Young tableaux into vacillating tableaux. Some insertions create horizontal steps and to preserve descents, these are bumped into lower paths.



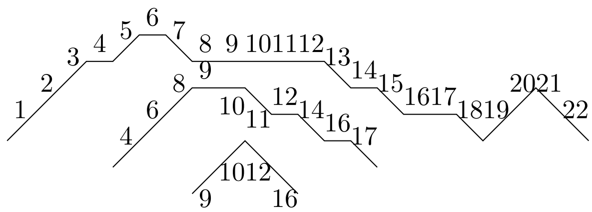
Algorithm 4: Vacillating Tableaux to Standard Young Tableaux

input : $n = 2k+1$, vacillating tableau V of dimension k, weight $\emptyset$, even length
output: standard Young tableau Q, same number of entries as V, n rows, all of even length

label V with $1, 2, \dots, r$ to obtain a labeled word w;
for $i = n, n-1, \dots, 2$ **do** /* extract row i */
- $j := \lfloor i/2 \rfloor$, unmark everything;
- **while** *in word j are $\lceil i/j \rceil$-row-positions* **do**
  - let p be the second position from left with $w(p) \in \{0, \pm j\}$, let $\tilde{p}$ be the next position left of p with $w(\tilde{p}) \in \{0, \pm j\}$;
  - $a = a_l = b = b_l = r$ for $l = 1, 2, \dots, j+1$;
  - **while** $p < b$ **do**
    - **if** *p height violation in l, for an $l < j$, ($p < a_l$ or not marked), if $b_{l+1} < p$ and $b_l = r$ ignore next l left of p* **then** /* height violation */
      - $w(p) := l$;
      - **if** $b_{l+1} = r$ **then** $a_{l+1} = r$ **else** $b_{l+1} = r$;
    - **if** *i is odd, $w(p) \in \{\pm j, 0\}$* **then** /* i odd */
      - **if** *$w(p) = 0$ on j-level* 0 **then** /* height violation special case */
        - $w(p) := j$, $b_{j+1} = r$;
      - **if** *p is 3-row-position, $p < b_{j+1}$* **then**
        - **if** *p is on level* 1 **then** /* mark separation point */
          - mark p and next 0 right, for $l \leq j$ mark $\pm l$ on l-level 0 between them;
        - **if** *$p < a_{j+1}$, right of p for no l there is an unmarked $\pm(l-1)$ between a marked $-l$ and a marked l and no -1 is marked* **then** /* adjust it */
          - change marked $\pm l$ into $\pm(l-1)$;
        - **else if** $a_{j+1} = r$ **then** $w(p) := -j$, $a_{j+1} := p$; /* a_{j+1} */
        - **else if** *p not marked, $b_{j+1} = r$* **then** $w(p) := -j$, $b_{j+1} := p$; /* b_{j+1} */
      - **else if** *$a_{j+1} < p$, $p < b_{j+1}$, $w(p), w(\tilde{p}) = 0, 0$, p j-even position, on j-level 1 for $a_j < p$ or 2 for $p < a_j$* **then** $w(p), w(\tilde{p}) := -j, j$; /* connect */
      - **else if** *$a_{j+1} < p$, $p < b_{j+1}$, $w(p), w(\tilde{p}) = -j, j$ on j-level 0* **then**
        - $w(p), w(\tilde{p}) = 0, 0$; /* separate */
    - **if** *i is even, $w(p) \in \{\pm j, 0\}$* **then** /* i even */
      - **if** *$w(p) = 0$ on $(j-1)$-level* 0 **then** /* height violation */
        - $w(p) := j-1$, $w(a_{j+1}) := j$, $a_{j+1} := a_j := r$; /* special case */
      - **if** *$a_j \leq p$, $b_{j+1} = r$* **then** $b_{j+1} := p$; /* b_{j+1} */
      - **if** *p is 2-row-position, $p < a_{j+1}$* **then**
        - **if** $w(p) = 0$ **then** /* mark separation point */
          - mark p and 0's directly right, for $l < j$ mark $\pm l$ on l-level 0 between them;
        - **if** *$p < a_{j+1}$, right of p for no l there is an unmarked $\pm(l-1)$ between a marked $-l$ and a marked l, there is not an unmarked $\pm(j-1)$ between two marked 0's and no -1 is marked* **then** /* adjust it */
          - change marked $\pm l$ into $\pm(l-1)$, change marked $0, 0$ into $-j, j$;
        - **else if** $a_{j+1} = r$ **then**
          - **if** $w(p) = 0$ **then** $a_{j+1} := \tilde{p}$, $w(\tilde{p}), w(p) := 0, -j$; /* a_{j+1} 1 */
          - **else** $a_{j+1} := p$, $w(p) := -j$; /* a_{j+1} 2 */
      - **else if** *$w(\tilde{p}), w(p) = 0, 0$ on j-level* 0 **then** /* connect */
        - $w(\tilde{p}), w(p) := j, -j$;

---

if $w(p) = -l$, $a_{l+1} < p$ *for an* l *with* $j \geq l > 1$, $b_l = r$ **then**
  if $a_l = r$ **then** $w(p) = -l + 1$, $a_l := p$; `/*` a_l `*/`
  else if $b_{l+1} < p$, $a_l < p$, *not marked* **then** $w(p) := -l + 1$, $b_l := p$; `/*` b_l `*/`
if $p = a_l$ *on* l*-level* 0, *marked* **then** mark the next l to the right; `/* mark` a_l `*/`
if $a_2 < p$, $w(p) = -1$, $a_1 = r$ **then** delete p, $a_1 = a := p$; `/*` a `*/`
else if $a_1 < p$, $b_2 < p$, $w(p) = -1$ **then** delete p, $b_1 = b := p$; `/*` b `*/`
let p be one position to the right, change $\tilde{p}$ according to it;
if $a \neq r$ **then** put a, b in row j of Q;
if i *is even* **then** change entries $\pm j$ of w into 0; `/* initialize` j `*/`
put the labels still in V in the first row of Q ; `/* extract row 1 */`
return Q;

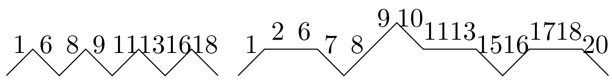


This description considers only the main cases of our Algorithm (in the algorithmic description these are those commented by: a_l, b_l, i *even* a_{j+1} *2*, i *odd* b_{j+1}, i *odd* a_{j+1}). The other cases are explained in the examples below.

Example 3.2.2 (A more complicated example to understand most common special cases). We consider the following tableau:

1	6	8	9	11	13	16	18
2	7	10	15	17	20		
3	12	14	21				
4	19						
5	22						

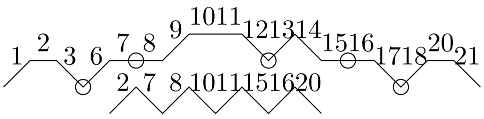
When we insert the first row, we see that our Algorithm is not descent-preserving and gives no sensible output if we follow our rules form the previous example strictly. Therefore we create another case for inserting something the first time into a path and change pairs of up-down-steps between them into horizontal steps. This refers to i *even* a_{j+1} *1* (at $2, 6$ and $17, 18$) and i *even connect* (at $11, 12$):



When inserting the third row we have to adjust this rules once again in order to preserve descents and that concatenated tableaux are mapped to concatenated paths. (A property that is only proven for $n = 3$, but conjectured otherwise, see Conjecture 3.4.1.) Therefore we have to introduce i *odd connect* and i *odd separate*. Between a and b two horizontal steps on level 1 are changed into a down-step and an up-step and a down-step and an up-step on level 0 are changed into two horizontal steps. (In our example we do i *odd separate* at $2, 6$, when inserting 3 and 12 and at $11, 13$ and $17, 18$ when inserting 14 and 21. We do i *odd connect* at $7, 8$ when inserting 3 and 12 and at $15, 16$ when inserting 14 and 21.) The corresponding positions are cycled.

Until this point our algorithm works exactly as in [3].

Now we initialize the second path. We see that where we did *separate* on our first path, there are some steps that do not observe the rules for vacillating tableau. We will deal with those in the next insertions.

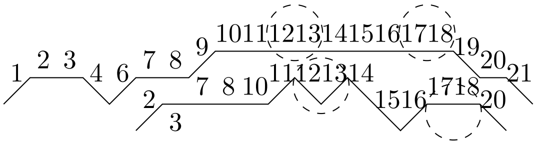


Now several things happen at once:

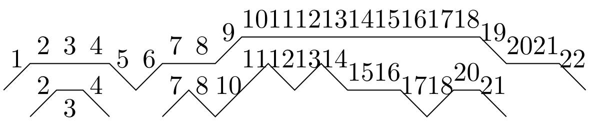
As mentioned above, we have to deal with this rule violations that we have noticed before. However we will see, that if a is inserted completely left of such a violation and b completely to the right, the rules we have introduced so far deal with that and two separate paths will be formed. We just mark them as "allowed hight violations" in *mark it + connect*. We do so between 3 and 6.

However between 17 and 18 we have to intervene and use *adjust separation point* (we see this at the right dashed circles). When doing so we ignore b_1 that is 19 and act as if 17 and 18 are still on level zero.

Moreover when inserting according to the simple rules we come to another point where the paths do not observe the rules of a vacillating tableau. Again we deal with this and use *height violation*, as this is not marked (we see this at the left dashed circles).



In the last insertion no new rule is introduced. However we can see how the two separate, concatenated paths have developed according to the two separate, concatenated tableaux our tableau consists of.

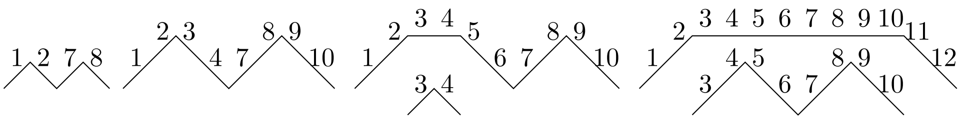


Note that this example is also an example for Algorithm 4, if one reads it from bottom to top.

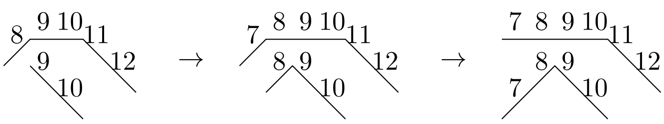
Example 3.2.3 (Illustrating additional special cases of Algorithm 3 and 4)*.* We consider several different tableaux to illustrate *ignore inserted* a_l in *height violation* (respectively *ignore next* l) and the two special cases of *height violation* i *even,* $a_{j+1} \neq 0$ and i *odd,* $w(p_j) = 0$ *on* j*-level* 0.

We start with the following tableau:

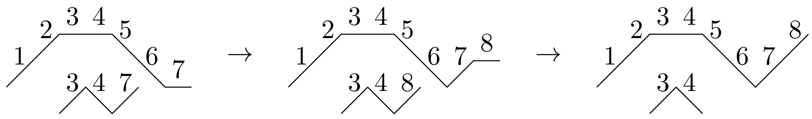
1	2	7	8
3	4	9	10
5	6		
11	12		



Inserting the first three rows works as before. However when inserting the fourth row we see in which cases we need *ignore inserted* a_l *if* $a_{l+1} = 0$ in *height violation.* When we insert $11, 12$, we have a *height violation* at $p = 8$. At $p = 7$ we have again a *height violation* as we ignore the inserted 11.



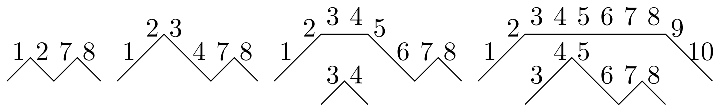
Now we consider the resulting vacillating tableau apply Algorithm 4. We get the labeled words in the opposite direction and obtain elements of our standard Young tableau two by two. We have a closer look at the first extraction as here again happens a special case. We extract $5, 6$ as a_2, b_2 and get a *height violation* at 7. We correct it and continue. At 8 we have again a *height violation*, as we ignore 7. Again we correct it and continue.



We point out, that this is also an example where a *height violation* of a overlaps with one of b. If this is not the case, then those special cases can also occur just in one of the two algorithms.

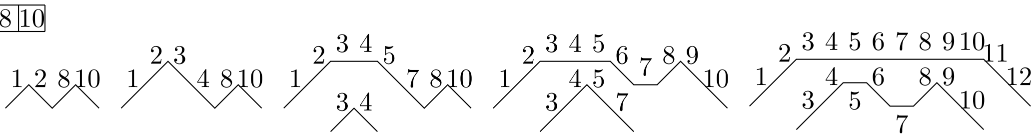
In the following Example, during insert row 4, we have to ignore 9 at $p = 7$ and get a *height violation* there.

1	2	7	8
3	4		
5	6		
9	10		



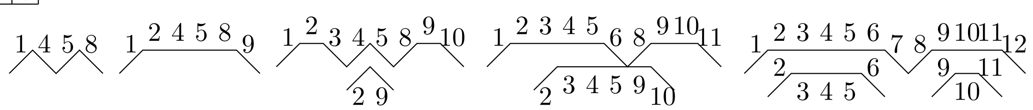
In the following Example, during extract row 5, we have to ignore 2 at $p = 8$ and get a hight violation there.

1	2	8	10
3	4		
5	7		
6	9		
11	12		

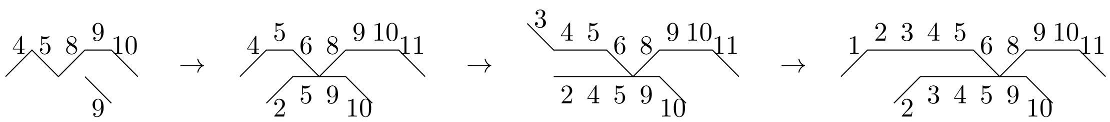


Next we consider the following tableau:

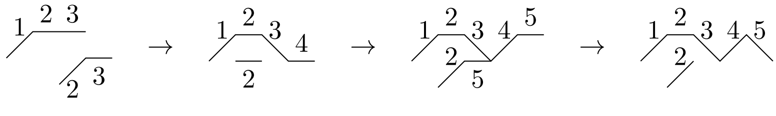
1	4	5	8
2	9		
3	10		
6	11		
7	12		



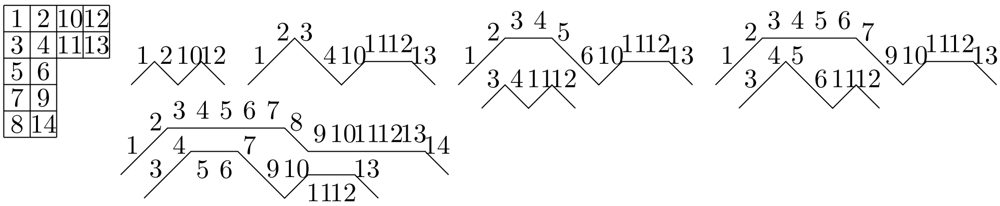
Again inserting the first three rows works as before. When inserting the fourth row, thus $6, 11$, we come into the special case of *height violation i even,* $a_{j+1} \neq 0$. The way we deal with this ensures, that we can continue normally after adjusting the *height violation*. At $p = 4$ we have a *height violation* that is marked but $p < a_2$. At this point a_3 is already defined to be 2. We set a_2 and a_3 back to 0 and search for them anew. Later we define them to be 3 and 2.



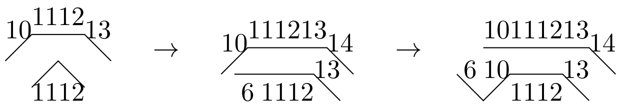
Again we consider the resulting vacillating tableau and apply Algorithm 4. Again we obtain the same sequence of labeled words but the other way around and extract elements of our standard Young tableau in pairs. We obtain our special case when extracting row 4, thus an even row. 4 is a 0 on 1-level 0. We set a_2 back to r and change 4 into a -1. Later we extract 5 as a new a_2.



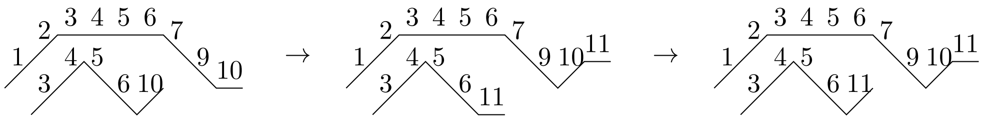
Finally we consider the following tableau:



This time inserting the first four rows works as before, however when inserting the fifth row we come the special case of *height violation i odd, $w(p_j) = 0$ on j-level* 0. Again the way we deal with this ensures, that we can continue normally after *adjusting the height violation.* This happens while inserting 8, 14. At $p = 10$ we have a *height violation*, where we had *i odd connect* at 6, 11. We insert b_2 again with 9.



Again we consider the resulting vacillating tableau and apply Algorithm 4. Again we obtain the same sequence of labeled words but the other way around and extract elements to our standard Young tableau in pairs. We extract 7, 8 for a_2, b_2 and 8 for a_1 and obtain a *height violation* at 10. We set $b_2 = r$ and correct it. Then we obtain a *height violation special case* at 11. This happens during extracting row 5, thus during *i odd.* We set $b_3 = r$ and $w(11) = 2$. We continue extracting 12 as b_3, 13 as b_2 and 14 as $b_1 = b$. Thus we extract 8, 14 as a, b.



Definition 3.2.4. *Separation points* are positions that are marked.

Example 3.2.5 (One tableau, different n)*.* In this example we consider a standard Young tableau Q with 7 rows and 2 columns in different dimension n. The first column of Q is filled with $1, 2, \ldots, 7$, the second column is filled with $8, 9, \ldots, 14$. As the rows have even length, empty rows are allowed. We see that *separation points* (positions that get marked) make a difference doing so, as those parts of the algorithms are the only ones executed for $a = b = 0$.

We see an illustration of this example in Figure 3.3.

When considering $n = 2k + 1 = 9$, we have $k = 3$ paths. When we consider $n = 9$ or $n = 11$ we see how *adjust separation points* alter the paths step by step and creates more paths. Finally we consider $n = 2k + 1 = 13$ and $n > 13$. We see that when going from 13 to 14 we add path 7, which is an up-step and a down-step. When considering larger n, the paths do not change anymore. Path 7 is dashed.

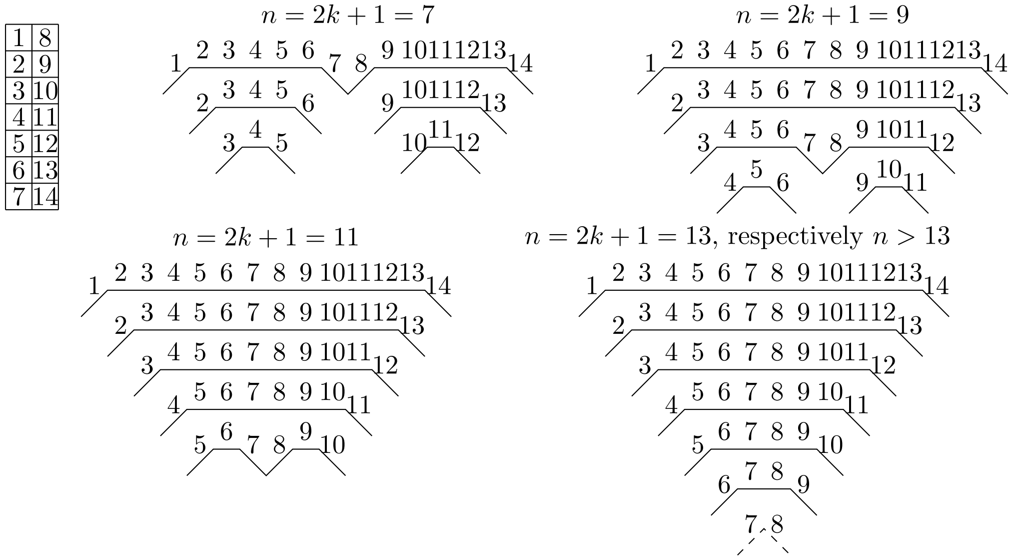
The reason for this phenomena is that 0's in a vacillating tableau are only allowed when the k-level is at least 1. Thus horizontal steps, that are truly horizontal steps, and not some other steps in paths below, are only allowed in the bottommost path.

This are the only differences when considering a tableau in different dimensions $n = 2k + 1$.

If we ignore everything not concerning j in Algorithm 3 and point out, that the combination of *separating* left of a_j and "change a_{j+1} into 0", corresponds to "insert a case 2" while insert the third row in [3], we get the following:

Theorem 3.2.6. *For tableaux in dimension three Algorithm 3 and Algorithm 4 generate the same output as Algorithm 11 and Algorithm 12 which are the according algorithms in [3].*

Figure 3.3: One tableau, different dimensions n



3.2.3 Properties and Proofs for Bijection B

The first goal of this subsection is to prove the following Theorem:

Theorem 3.2.7. *Algorithm 3 is well-defined and produces a vacillating tableau V of length r and dimension k given a standard Young tableau Q with $2k+1$ rows of even length and r entries.*

We prepare the proof by stating and proving several lemmas concerning Algorithm 3. We use notation form Algorithm 3. Variables, etc. also refer to it. Moreover we call marked positions *separation points*.

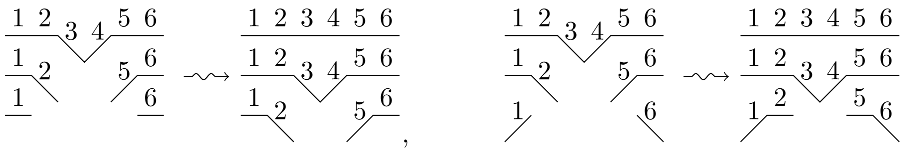
Corollary 3.2.8. *For j even, we redefine a_{j+1} to be the next unchanged j so far to the left of a_j and b_{j+1} of be the next changed position to the left during the insertion process of a and b so far. This is just a renaming. However it follows that the l-level grows between c_l and c_{l+1}, but not somewhere else.*

Corollary 3.2.9. Adjust separation point *changes the marked positions as if an a was inserted in between and a b was inserted to the left.*

Lemma 3.2.10. Separation points *contain* height violations *exactly after initializing a new j and after inserting an even row. This* height violation *is always in $j-1$. This implies that they cause no* height violation *in the end of our insertion process.*

Proof. Separation points always start at *separate odd* between a and b_{j+1}. As long as they are still between such newly inserted elements, they expand on j. After inserting an odd row there is always another *separate odd* and thus at the last inserted odd row there is no *height violation* anymore.

Once we come into the case *adjust separation point* we do the same as some inserted a, b would do. However one path after the other, beginning with the topmost, is not part of the *separation point* anymore.

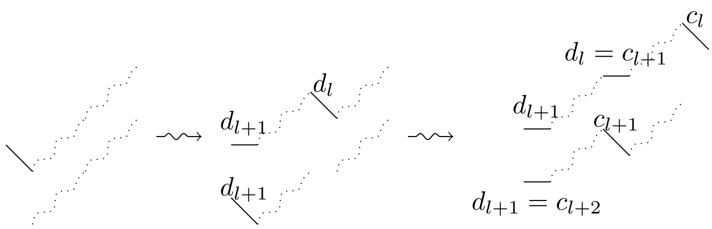


□

Definition 3.2.11. The predecessor of some number a or b, namely c, is the following number d. Search for the number $c = c_1$ that is inserted first during the insertion process. This number c has some other number directly below in Q, namely d. We refer to its insertions with d_l like we do for a or b Algorithm 3.

Lemma 3.2.12. *If an inserted number c_{l+1} equals its predecessor d_l, no new* height violation *arises.*

Proof. Inserting d_l made the l-level higher between d_l and d_{l+1} and did not change the $(l+1)$-level in this area. When choosing d_l as c_{l+1} this makes the $(l+1)$-level higher in the same area or a smaller one, (and might changes $\pm l$ into 0's with a level grow of 1) which can cause no *height violation*. The following sketch illustrates this for $l + 1 \neq j$.



□

Lemma 3.2.13. *1. If p is a* height violation *in $(l-1)$, $w(p)$ is always $(l-1)$.*

2. There are no height violations *after inserting a pair of a, b if there where no before. The only exceptions are* separation points *where there is a* height violation *in $(j-1)$ before and after inserting an even row.*

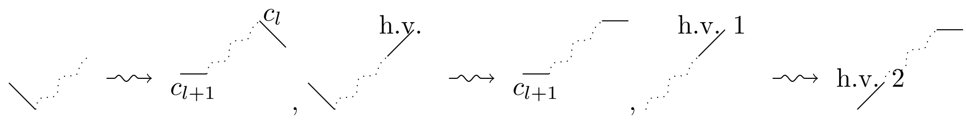
3. We can always find a_{j+1} and b_{j+1}.

Proof. We show these three statements inductively. The base case is clear (empty case). For the inductive step we show one statement after the other.

1. If there was no *height violation* before, to find a new one, we consider in which situations the level in some paths can grow:

   - between c_l and the first c_{l+1};
   - between a *height violation* in l and the new c_{l+1};
   - between a *height violation* in l and a new *height violation* in $(l-1)$.

   In this area cannot be a $-l$ except a marked one or a_l if c is a b, as this would have been taken for c_{l+1} otherwise. We illustrated these three cases for $l + 1 \neq j$.

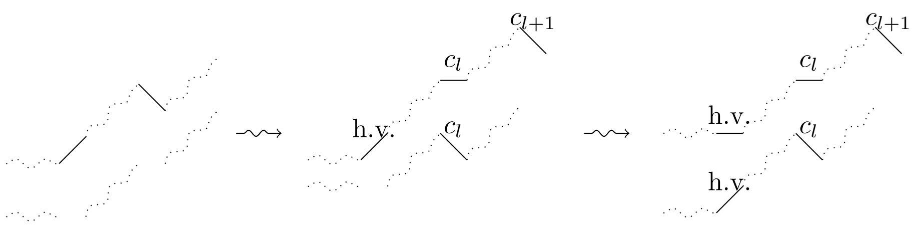


- *i odd connect* or *i even connect*;

  This happens between a and b and can only create a *height violation* at a position with j-level 0, thus a $j-1$. ($-j+1$ could also create a *height violation* but needs to be left of such a $j-1$). At *i even connect* such positions are marked.

Height violations in $(l-1)$ only happen in those situations. p cannot be a 0 or else the previous p would be a *height violation* as well and we see inductively that right of p there are no *height violations*. The same argument holds for l and $-(l-1)$. We have seen that in the area where word l gets higher there is either no $-l$ or it is ignored (a_l) or it is to the left of a $l-1$ (which is marked). Therefore the only possible value is $l-1$.

2. Therefore at a *height violation*, the $(l-1)$-level increases and the l-level decreases. However as this happened somewhere where the l-level was increased before (by inserting c_l), it sets the l-level to its original height. Thus there is no *height violation* until another level growth.



Height violations can only add up in pairs of two (*connect* only happens left of b_{j+1}). If that happens, also the levels of that *height violations* add up, so it is sufficient to look at each situation separately. The *ignore* a_l ensures that everything is considered separately. (Compare with the first tableau in Example 3.2.3.)

3. To conclude we note that once we reach the predecessor no new *height violation* arises (compare with Lemma 3.2.12). This also implies inductively that the predecessor cannot be taken from a pair before, as their predecessors are the leftmost positions they can take.

   It remains to show that the predecessor d_l can be taken indeed as new c_{l+1}. This holds for $l+1 \neq j$ as in this case $d_l = -l$. For $l+1 = j$ we distinguish between i even and i odd.

   If i is odd we can argue the same except for the case that $d_l = 0$. In this case, however, *i odd separate* changes this d_l into a $-l$.

   If i is even we can argue that in *i odd* a's produce 0's that are in 0-even positions and b's produce 0's that are in 0-odd positions. Now a_{j+1}'s are always j-even positions and b_{j+1}'s are always j-odd positions and every such position can be chosen as such. □

Remark 3.2.14. Left sides (down-steps) of *separation points* can never be inserted as b thus are never predecessors of b.

Lemma 3.2.15. *The sum over the labeled word w is 0 after every insertion of a pair a, b. In particular the sum over $\pm l$ in w is 0 for every l.*

Proof. The sum over all j's is 0 after initializing j as there is always an even number of 0's. Thus we have to show that nothing we do during the insertion process, changes the sum:

- If i is even and we insert a_j, a_{j+1} and b_j, b_{j+1} we insert either two $-j$'s and change one $-j$ into a j or we insert one $-j$ and one 0 and change one $-j$ into a 0. Otherwise we insert something with $-l$ and change another $-l$ into a 0 or a $(-l+1)$.

- When *i even connect*, *i odd connect* or *adjust separating point* we always change an l and a $-l$ into 0's or into $l+1$ and $-l-1$. At *i odd separate* we change two 0's into a $-l$ and l.
- At *height violation* we do the inverse of finding a c_l, namely changing a $l-1$ into an l instead of changing an $-l-1$ into an $-l$. As we insert c_l anew later on, this does not change the sum. However, there are two situations where it could be that c_{l+1} is already found but there is still a *height violation*. Therefore we have to adjust the path to ensure sum 0 in this situations. Those are the two special cases. At *i odd connect* we deal with this by changing a 0 into a $-j = -l-1$ again. At *i even* a_{j+1} *1* we defined and adjusted a_{j+1} before and deal with this by changing it back again. □

Proof of Theorem 3.2.7. For well-definedness, we have to show that the while loop always terminates, thus that we find an a_{j+1}. We have seen this in Lemma 3.2.13.

Moreover we have to show that the vacillating tableau properties hold for our resulting word:

1. In every initial segment the following holds:
   (a) $\#i - \#(-i) \geq 0$,
   (b) $\#i - \#(-i) \geq \#(i+1) - \#(-i-1)$,
   (c) if the last position is 0 then $\#k - \#(-k) > 0$.
2. The sum over all positions is 0.

To show that Property 1a is satisfied after any insertion of a pair a, b, we have to show that there are no steps with negative l-level. There are two steps in the algorithm where we decrease the level of some position. At the first one, *separate odd*, we generate $-j, j$ on j-level one. At the second one, *height violation*, we have seen in Lemma 3.2.13 that we decrease positions that have been increased before.

To show that Property 1b is satisfied, we have to show that there is no *height violation*. This is shown in Lemma 3.2.13 and Lemma 3.2.10.

To show that Property 1c is satisfied we show that 0's are always at least on k-level one. When initializing a new j, 0's get changed into $\pm j$. New 0's come either from *connect*, where they are on level one or at c_{j+1}, *i odd*, where we change a $-j$ on level at least 0 to a 0 with level at least one or more.

Property 2 is shown in Lemma 3.2.15.

Finally, the number of steps is r as every entry of Q inserts exactly one step. □

Due to what we have seen about the predecessor, the following lemma holds for even length paths and standard Young tableau with all rows of even length.

Lemma 3.2.16. *Considering Algorithm 3, concatenation of vacillating tableaux of empty shape and even length corresponds to concatenation of standard Young tableaux whose rows have even length.*

In particular, the following holds:

- *If a vacillating tableau is composed of two concatenated paths of empty shape and even length, its corresponding standard Young tableau can be written as concatenation of two standard Young tableaux all whose rows have even length.*
- *On the other hand if a standard Young tableau can be written as concatenation of two standard Young tableaux whose rows have even length, its corresponding vacillating tableau is also composed of two concatenated paths of empty shape and even length.*

Now we want to show the same for Algorithm 4 which we will prove later to be the reversed algorithm of Algorithm 3. To see this we will again provide and prove several lemmas first.

Theorem 3.2.17. *Algorithm 4 is well-defined and produces a standard Young tableau, with rows of even length and r entries, given a vacillating tableau of even length r and empty shape.*

Lemma 3.2.18. *The sum over positions in the labeled word w is 0 after any extraction of a pair a, b. In particular the sum over $\pm l$ in w stays 0.*

Proof. For every c_l that is changed from $-l$ into $-(l-1)$ we change a $-(l-1)$ into a $-(l-2)$. If we consider c_j, i odd, we loose a 0 in this process. If we consider c_j, i even, we conclude that we either extract two $-j$'s and change a j into a $-j$ or we extract one 0 and one $-j$ and change a 0 into a $-j$. If we consider 1 we simply delete a -1 after we inserted a new one to the left.

Connect, *separate*, *height violations* and *separation points* also just change l in pairs of two - always an l and a $-l$. For details compare this with the proof of Lemma 3.2.15. □

Lemma 3.2.19. *If p is a* height violation *in l, p is always an $l+1$. After extracting a pair of a, b there is no* height violation *if there was no before and the extraction process stops. Again the only exceptions are* separation points*.*

Proof. Once again we consider a and b separately as combined *height violations* just add up. *Ignore positions that are corrected height violations of a* ensures that everything is considered separately. (Again, compare with the first tableau in Example 3.2.3.)

For a *height violation* we have to consider where the l-level for $l > j$ is decreased:

- between c_{l+1} and c_l;
- between c_{l+1} and a *height violation* in $l+1$;
- when *adjusting a height violation* in $l-1$ until finding a new c_l or a new *height violation*;

In this area cannot be a $-l$ except for a marked one, if c is a b, as this would have been taken for c_l.

In the proof of Lemma 3.2.13 those situations are illustrated for Algorithm 1. Here the situation is similar but the other way around.

Moreover we have consider level increasings. Those are only possible for the j-level by *separation odd*. However this happens at j-even positions, so if there is something on j-level zero we change it into $j-1$ and set b_{j+1} and b_j to undefined in *height violations special cases*. Thus there is no *height violation* afterwards.

An l is not a *height violation*, as there would have been be one before. The same holds for a $-(l+1)$. Moreover it cannot be a $-l$ as we have seen in the list above, thus it is an $l+1$. At *adjust height violations* this $l+1$ is changed into an l, thus we increase the l-level and decrease the $(l+1)$-level. However this happens somewhere, where the $(l+1)$-level has been increased before by choosing c_{l+1}.

Again the illustrations in the proof of Lemma 3.2.13 show the same situations in Algorithm 3. □

Lemma 3.2.20. *We always find an a_l and a b_l if we found an a_{l+1} and a b_{l+1}. After several steps we find an a_l and a b_l whose extraction does not cause a new* height violation*.*

Proof. We start with a_{j+1} and b_{j+1} and distinguish the parity of i.

If i is odd, a_{j+1} changes a 0 in 3-row-position. (If none exists anymore due to *adjust separation point*, nothing happens.) This is the only time when a 0 is changed except for b_{j+1} and *connect/separate odd*, where 0's are changed in pairs of two. Thus after finding a_{j+1} there will remain an odd number of 0's, therefore we will find a b_{j+1}.

If i is even, we only need to find a_{j+1} as this gives us both a first a_j and a first b_j (a_j is a 0 or a $-j$ on j-level at least one, thus we find b_j as a $-j$ on j-level at least zero). We find it left of a 2-row-position. Again if none exists anymore, nothing happens.

For $l < j+1$ we always find a first c_l as c_{l+1} used to be a 0 not on level zero, so there is a $-l$ to the right. After a *height violation*, we insert an l and get the old l-level back so we can find another $-l$. If there would be only one $-l$ for both a and b, that would mean, that both a_{l+1} and b_{l+1} were $-(l+1)$ on l-level zero with no $-l$ in between. However this would either cause a *height violation* in a_{l+1}, which is a contradiction or there is an $(l+1)$ on $(l+1)$-level 0 next to a_{l+1} and $b_{l+1}, \dots, b_j$ are right of that. Moreover $a_{l+1}, \dots, a_j$ are on level 0 each. As there is no $-l$ between a_{l+1} and this $(l+1)$ a_j satisfies the conditions for *adjust separation point* as there is no l in between either (b_{l+1} is also on level 0), which is also a contradiction.

We find a c_l whose extraction does not cause a new *height violation* as this is the case once we reach a $-l$ not followed by an l. This is the case at some point in a path without *height violations*, that ends on l-level zero for all l due to Lemma 3.2.18 and Lemma 3.2.19. □

Lemma 3.2.21. *For each extracted number in row $i+1$ we extract at least one smaller number in row i.*

Proof. We show first that that every last d_{l+1} can be a last c_l for $l \neq j$: Extracting d_{l+1} decreased the l-level by 1 for $l < j$ between d_{l+1} and d_l without changing the $(l-1)$-level there. Thus we can decrease the $(l-1)$-level in this area when extracting the next row, thus this could be a c_l that causes no further *height violations*. (Compare with the illustrations in the proof of Lemma 3.2.12.)

Now we show that each extracted d_j causes a c_{j+1}. Again we do so by distinguishing the parity of i when extracting c_j.

If i is odd, we extracted $-(j+1)$'s before in an even process. We need to show that those produced 0's in 3-row-positions and that we can take those as a_{j+1} and b_{j+1}. We distinguish two cases. If those are separated by a_j, they produce automatically two odd sequences of 0's, one of which to take. It could be the case that one of them gets even due to some former or later c_j in this round, however this is the same situation as in the next case. If those are not separated by a_j, there cannot be a $-j$ between those. Thus at least the right one is on j-level two or higher and the other one is either on the same level, or if separated by a j, it is in an odd sequence.

If i is even, we extracted j's in an i-odd-process before. Thus, due to Corollary 3.2.9 we can use Lemma 34 and its proof in [3].

It remains to show that those new 2- or 3-row-positions will not be changed in *adjust separating points*. This follows as the extracting process of a will leave some negative step in between or will extract a -1. □

Proof of Theorem 3.2.17. For well-definedness we have to show that the two while loops terminate. That the inner one terminates ensure Lemma 3.2.19 and Lemma 3.2.20. The outer one has to terminate, as with each extraction the word gets smaller by two, so in the end there is nothing left to build a 2-or 3-row position.

Using Lemma 3.2.21, we see that it produces a standard Young tableau with even row lengths. □

Theorem 3.2.22. *Algorithms 3 and 4 are inverse.*

Proof. We show this by showing that every step has its inverse in the other algorithm. Those steps are named (commented) the same. We consider them separately.

For the following steps it follows directly from the definition that they are inverse:

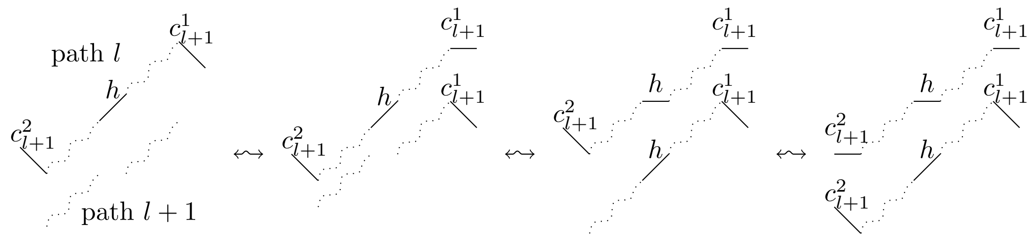
- Initialize j
- Insert / extract row 1
- a / b getting inserted or extracted as a_1 / b_1

For the following steps we have to argue a little more:

- *Height violations:*

  We show that if a *height violation* h with $w(h) = l$ in Algorithm 3 occurs after inserting c^1_{l+1} and we correct it, and insert c^2_{l+1} later, we get the same h with $w(h) = l+1$ as *height violation* when extracting this c^2_{l+1}, and the other way around. This is sufficient as due to the fact that *height violations* in l are always l respectively $l+1$, they act inverse.

  When we *adjust* a *height violation* h in Algorithm 3, we get a $w(h) = l+1$ whose $(l+1)$-level is one less than its l-level. When extracting c^2_{l+1} in Algorithm 4, this decreases the l-level to the right, and h is the first position with an l-level that is too large, as the other positions were no *height violation* before starting *height violation* in Algorithm 3. The other way around is similar. If we find a *height violation* $w(h) = l+1$ caused by extracting c^2_{l+1} in Alorithm 4 we change this into an l and extract a c^1_{l+1} to the right. When inserting c^1_{l+1} in Algorithm 3, we increase the $(l+1)$-level such that exactly at h there is a new *height violation*. Again nothing earlier could have caused it as those positions were no *height violations* in Algorithm 4. We illustrated this for $l+1 \neq j$ in the following sketch.

  The three special cases are left to consider.

  Ignore a_l at a *height violation* happens when a_l is already inserted but a_{l+1} is not, thus the level of path l is changed but the level of path $l+1$ is not. When we do the inverse this is not relevant as in this case a is always extracted further than b. Thus we have to do the same if b_{l+1} is extracted but b_l is not. Therefore this special case changes nothing from the argumentation above.

  The special case i *even*, happens if there is a *height violation* between a_{j+1} and a_j. Thus, together with finding a new a_j, it sets $-(j-1), (j-1)$ between those to $0, 0$. Therefore when we extract we get the left 0 as a_j and change it into $-(j-1)$ again. The special case changes the right 0 into $(j-1)$ and sets a_j and a_{j+1} to undefined. For an illustration of an example see the fourth tableau of Example 3.2.3. The other direction works the same way.

  The special case i *odd* happens after *connect*. Thus it changes a $-j$ changed into a 0 back into a $-j$. Then b_j is inserted anew as this $-j$. When it gets extracted, it detects *height violations*. Correcting them leaves the 0 we produced on j-level 0, which we change back into a j. For an illustration of an example see the fifth tableau of Example 3.2.3. The other direction works the same way. We point out that Algorithm 1 *connects* between a_{j+1} and b_{j+1} at j-level 0 whenever i is odd.

- *Separation points:*

  In both algorithms we *mark* and *adjust separation points* while searching for a's and b's. This way we *adjust separation points* before reaching the next a and b. Thus in Algorithm 3 this happens right of a and b and in Algorithm 4 this happens to the left. This makes no difference as we consider everything still in the same order, and make an extra iteration for *separating points* at the ends not considered so far.

  We mark positions $\pm l$ at certain points, to make certain exceptions for them. We mark 0's also (in a slightly different way), but those are not relevant as those are never such exceptions.

  The *separation points* we just mark are for an l between a_l and b_{j+1} and mark positions $\pm l$ up to $\pm j$. Due to *i odd separate* and *i even connect*, we mark in each algorithm positions that way that they form the same pattern after the iteration as the other algorithms marks in the beginning of an iteration. The marking a_l ensures that even though a might be inserted on the left part of the *separation point*, all $\pm l$ that should be marked are marked.

  The *separation points* we *mark* and *adjust* in Algorithm 3 are for no l between a_l and b_{j+1}. When we *adjust a separation point* in Algorithm 3, it is automatically right of the current b_l. Thus, what we have to show is that exactly *separation points* we *adjust* form the patterns we demand in Algorithm 4 *adjust separation points.*

  When an a is in between there are three different ways it can be so. When a is inserted as such, we have a marked -1. When a starts to be in between in the first path marked, lets call it l, then a_{l-1} is either between the marked positions $\pm l$, therefore a $-(l-1)$ is between some marked positions or it is not, thus it changed a marked $-(l-1)$ into a $-l$ but not the according $(l-1)$. When a causes a *height violation* even though it is marked and therefore $p < a_l$, we can argue as above. This explains why we look for $\pm l$ between the 0's directly to the right. We do so because in Algorithm 3 we mark all $\pm l$ between a $-j$ until the next j, which become the leftmost and rightmost 0 of their sequence of 0's.

  When we *adjust a separation point*, we shift the $\pm l$ upwards, as $-(l-1)$ was not between $\pm(l-1)$ it is now not between $\pm l$ any more.

With the knowledge of those, the following gets easier:

- *i even, i odd*:

  It remains to show that everything that does not involve marking is inverse.

  We first show *i odd* produces 0's in 3-row-position while *i even* produces 0's in 2-row-position.

  We consider *i even* first. There are two possible ways to produce 0's when inserting the second row: at "inserting a case 1", and changing pairs $1, -1$ into $0, 0$. Both produce them in pairs on j-level one. Moreover, those are always right of the current a_j. As the next insertion happens strictly left of the next b_j, that is itself left of a_j, and 0's from separation points are inserted in pairs strictly between those, they cannot be isolated later by inserting the second row.

  Now we consider *i odd*. Elements are inserted decreasingly, from right to left. Therefore left of the leftmost a_{j+1} so far there are no 0's in 3-row-position. Moreover, "separate" and "connect" only changes or produces pairs such that an even number of positions in $\{0, \pm j\}$ and thus an even number of 0's is to the left. Now we consider a_{j+1} and b_{j+1} of a pair to insert. If those are on j-level two or higher, they are 0's in 3-row-position. If b_{j+1} is a 0 on j-level one, it needs to be between a_j and b_j. If a_j is the $-j$ to end the sequence of

0's of b_j it could be the case that this sequence is even. In this case, the leftmost 0 of this sequence is at j-level two. Moreover a_{j+1} is either another 0 on j-level two or a j next to an odd number of 0's on j-level one, as it deleted one. Now a_{j+1} either produced a new 0 or it deleted one. In the former case, if this is on j-level one, it needs to be either a single 0 or part of a sequence that was there before. Thus, that sequence was even before adding a 0. In the latter case this deleted a 0 that was in an even sequence of 0's, hence is now a new odd sequence of 0's.

Now Algorithm 4 extracts elements into an odd row only as long as there are 0's in 3-row-position and elements into the next even row as long as there are 0's in 2-row position. As insert and extract are inverse there is nothing left to show.

- a_l / b_l:

  We point out, that we have an index shift of one at l between the formulations. Once we consider this, we see that they operate clearly in the opposite way. It remains to show, that they act on the same positions. As they always take the next $-l$, and change it, there is no $-l$ that could be taken before from the other algorithm. □

Theorem 3.2.23. *Algorithm 3 is descent preserving.*

Proof. We show that the algorithm preserves descents after every insertion of a pair a, b in the sense that we consider the inserted numbers as a new total order.

In the first step we show that when we insert a pair (a, b), a and b cause a descent except for the case that a and b are neighbors in the order of already inserted numbers. To see why we want this to hold, we consider the partial standard Young tableau consisting only of already inserted numbers. The number smaller than a needs to be in a row below a as numbers in the same row to the right are larger. The same holds for b except if a and b are neighbors in the current order.

In the case that a and b are neighbors, they are both inserted as -1's and we have to show that only a is a descent. Thus everything else is analogously to the general case.

As a or b are inserted with -1 the only way that this causes no descent is that the position to the left is a -1 or a 1 on level zero. The latter is not possible as this 1 would have been on level -1 before. A -1 directly to the left of a or b would change either into a 0, a 1 or a -2, depending on i. All cases cause a descent.

In the second step we show that we do not lose descents when inserting a pair (a, b). If an entry was a descent in the partial tableau before inserting (a, b) it is still one in the new partial tableau, either with the same number above, or with a or b. In the former case neither a nor b are inserted between those. In the latter case either a or b is inserted in between. This creates a descent in the vacillating tableau and removes the other descent as $(-1, x)$ can never be a descent.

Inserting a $-l$, always creates a new descent, when ignoring positions with smaller absolute values. The only such value that is not a descent left of a $-l$ is $-l$. However a $-l$ left of an inserted $-l$ is changed into a $-l-1$ while inserting. (Separation points are ignored if $c = b$, however they are adjusted before, if b would be inserted between those, thus changed into a $-l-1$ too.)

It follows, that the position left of our new $-l$ is a descent if and only if it was a descent before changing it into $-l$.

In the third step we consider *connect* and *separate* as well as *height violation* and *adjust separation points*:

We show that *separate* and *connect* neither produce nor cancel a descent. For i *even connect* this is clear as $(j, -j)$ on l-level zero is not a descent. For i *odd separate* we consider a 0 left or right of a position that was changed in *separate*. Those need to be either $\tilde{a}$ or $\tilde{b}$ or they were changed in *connect*, because otherwise they would have been *separated* also. In the former case we want a descent, in the latter too, as $(j, 0)$ has changed into $(0, -j)$ or the other way around. The same holds for j's or $-j$'s of *connect*.

At *height violation* we change an l that was a *height violation* to an $(l+1)$. If l was a descent, and $(l+1)$ is none, there needs to be a $(l+1)$ to the right, however this cannot happen, as then the *height violation* would have started earlier to the right. If l was no descent, then $(l+1)$ is none too. In our special cases we undo some change we have just done before, thus we do not change any descents.

For *separation points* we point out that a descent left of a $\pm l$ needs to be a $\pm(l+1)$, thus when *adjusting* them they get either $\pm(l+1)$ and $\pm(l+2)$ or $\pm(j-1)$ and 0, both preserves the descent. □

Theorem 3.2.24. *Let Q be a standard Young tableau with rows of even length and V be its corresponding vacillating tableau determined by Algorithms 3 and 4. If and only if for all rows $i = 1, 2, \ldots, 2k+1$ of Q the first position in row i is i, the first k steps of V are $1, 2, \ldots, k, 0, -k, -k+1, \ldots, -1$.*

Proof. This holds as Algorithm 3 is descent preserving and Algorithms 3 and 4 are inverse. □

3.3 μ-horizontal strips

In this section we will consider μ-horizontal strips, which are the link between our two bijections. A alternative orthogonal Littlewood-Richardson tableau $L \in \mathrm{aLR}^{\mu}_{\lambda}$ together with a standard Young tableau $Q \in \mathrm{SYT}$ define a bigger standard Young tableau $\tilde{Q}$ containing a μ-horizontal strip. To obtain it we add entries to Q to those spots where there are numbers in L. We fill the spots increasingly from left to right, starting with the biggest numbers in L.

On the other hand we are interested in the shape of the corresponding vacillating tableau. We want it to have shape μ once we deleted the last $|\mu|$ entries. Therefore we also introduce a structure of vacillating tableaux we call "cut-away-shape".

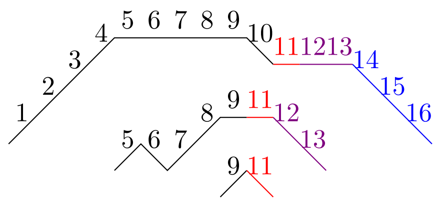
We start this section by defining the structures we use and giving some examples. After that, we will show that standard Young tableaux containing a μ-horizontal strip are mapped to vacillating tableaux of cut-away-shape μ. With this we complete proving the properties in question for our bijection.

Definition 3.3.1. A vacillating tableau of shape $\emptyset$ has *cut-away-shape* $\mu = (\mu_1, \mu_2, \ldots, \mu_l)$ if it ends with

$$(\underbrace{-l, -l, \ldots, -l,}_{\mu_l} \quad \underbrace{-l+1, -l+1, \ldots, -l+1,}_{\mu_{l-1}} \quad \underset{\cdots}{\cdots} \quad \underbrace{-2, -2, \ldots, -2,}_{\mu_2} \quad \underbrace{-1, -1, \ldots, -1}_{\mu_1}) \;.$$

Therefore, if we delete "cut away" the last $|\mu|$ positions the vacillating tableau has shape μ.

Example 3.3.2. The following vacillating tableau has cut-away-shape $\mu = (3, 2, 1)$:



Remark 3.3.3. If a tableau has cut-away-shape $\mu = (\mu_1, \mu_2, \dots, \mu_l)$ it has also cut-away-shape $\tilde{\mu}$ where $\tilde{\mu} \subseteq \mu$ are subpartitions of the form $\tilde{\mu} = (\mu_1, \mu_2, \dots, \mu_m, \mu_{m+1} - u)$ for every $0 \leq m < l$ and $0 \leq u < \mu_{m+1}$.

Definition 3.3.4. Let μ be a partition with $\ell(\mu) \leq k$. Let Q be a standard Young tableau with $2k+1$, possibly empty, rows, whose lengths have all the same parity, and r entries. A *μ-horizontal strip* is a pattern of the last $|\mu|$ numbers in the following way:

1. For each j, the numbers $r - (\mu_1 + \mu_2 + \dots + \mu_{j-1}) - \mu_j + 1$ up to $r - (\mu_1 + \mu_2 + \dots + \mu_{j-1})$ form a horizontal strip filled increasingly from left to right.

   By abuse of notation we say that those numbers are in μ_j.

2. The ith number in μ_j is in a row below the ith number of μ_{j+1} if the latter exists.

3. Go through the elements of Q belonging to the $|\mu|$ last numbers from top to bottom, from right to left. Let e be the current element of the μ-horizontal strip. We define a sequence v_e of elements of the μ-horizontal strip. Let e be the first entry of v_e. If $m-1$ entries of v_e are defined, let f be entry number $m-1$. We search now for entry number m. For that we consider entries whose that are smaller than f and which are in exactly $m-1$ sequences defined before v_e. If this set is nonempty, take the largest entry as entry m. If it is empty, v_e has no more entries.

   Let r_e be the row p is in. Now we define the value o_e to be the number of entries in v_e with the following properties. It is the rightmost occurrence in their μ_j and if number m in v_e, all $v_{\tilde{e}}$, where $\tilde{e} \neq e$ is in the same row as e, have at most $m-1$ entries.

   We require $r_e \geq 2|v_e| - o_e$.

Proposition 3.3.5. *If and only if the $|\mu|$ largest elements in a standard Young tableau Q form a μ-horizontal strip, the reverse skew semistandard tableau we obtain by deleting smaller elements and replacing elements in μ_j by j is an alternative orthogonal Littlewood-Richardson tableau.*

Proof. This follows directly from the definitions (Definition 2.2.1 and Definition 3.3.4). The main difference in the definitions is that in the μ-horizontal strip we only require in the third point of defining v that it is the largest one, and not that it is the rightmost occurrence. Since entries in μ_j are increasing, this is still equivalent. ☐

Example 3.3.6. We consider the following tableaux (the first and the last one are corresponding tableaux to those in Example 2.2.2):

1	2	3	4
5	6	7	8
9	11	13	16
10	12		
14	15		

1	2
3	4
5	6

1	2	5	6
3	4	7	10
8	9		

1	2	3	4
5	6	11	12
7	8	17	18
9	10		
13	14		
15	16		

1	2	3	4	21	22	23	24
5	6	7	8	25	26	27	28
9	10	11	12	29	30	32	33
13	14	15	16	31	35	36	37
17	18	19	20	34	42		
38	39	40	41				

The first tableau contains a $\mu = (3,2,1)$-horizontal strip (as well as a $(3,2)$-, $(3,1)$-, (3)-, (2)-, (1)-, and $\emptyset$-horizontal strip). It is the corresponding standard Young tableau to the vacillating tableau in the previous example. The v's are: (16), (13), (11), $(12,11)$, $(15,13,11)$, $(14,12)$. (Compare with Example 2.2.2.)

The second tableau contains a $(2,1)$-horizontal strip but not a $(2,2)$, $(2,2,1)$ or $(2,2,2)$ one due to the third condition. The v's are: (4), $(6,4)$, (5).

The third tableau contains a $(3,1)$-horizontal strip. The v's are: (10), (7), $(9,7)$, (8).

The fourth tableau contains a $(4,2,1)$-horizontal strip but not a $(4,2,2)$-horizontal strip due to the third condition. The v's are: (12), $(18,12)$, (17), (14), (13), $(16,14,12)$, $(15,13)$.

The last (fifth) tableau contains a $\mu = (5,4,3)$-horizontal strip. The v's are: (33), (32); (37, 33), (36, 32), (35), (31); (42, 37, 33), (34, 31); (41, 36, 32), (40, 35, 31), (39, 34), (38).

Before we prove that μ-horizontal strips are equivalent to cut-away-shapes, we state some facts about Algorithm 3 we will need later on. These follow directly of the formulation of the algorithm. We see that everything happens right of the rightmost up-step that is not part of the right part of a *separation point*. Therefore *height violations* do not play a role here.

Proposition 3.3.7. *For the $|\mu|$ largest positions in Q it holds that:*

- *A $-l$ gets a $-(l+1)$ if and only if it is chosen as some c_{l+1} for $l < j$.*
- *$-j$'s chosen get $(j+1)$ or $-(j+1)$ when chosen as c_{j+1} in insert row $2j+1$. (They get 0's first, and are initialized later.)*
- *$-j$'s chosen get either 0 or j when chosen as c_{j+1} in insert row $2j$. Only if there is only one position right of them they become a 0 still in our considered part of the path.*
- *An l can only get a negative entry if it is part of a* separation point.

Lemma 3.3.8. *We consider an element e in μ_i in row r_e which gets inserted.*

1. *For $l \leq \lfloor r_e/2 \rfloor$ element number l in v_e is e_l.*
2. *If $|v_e| < \lfloor r_e/2 \rfloor$ e_l with $l > |v_e|$ are left of the part of the labeled word we consider.*
3. *If $|v_e| > \lfloor r_e/2 \rfloor$ element number l with $l > \lfloor r_e/2 \rfloor$ is part of a* separation point *directly left of our down-steps. Each time nothing is changed to the right of it, the rightmost one of those gets a $-j$.*

Proof. We prove this inductively on the row r_e an element is in.

For the base case we consider an element of the first row. This is the only one belonging to the μ-horizontal strip and the last one of the first row. Thus it gets inserted as a -1. One could say that it was inserted as a 0, thus part of a *separation point*, but changes into a $-j = -1$ when *initializing row* $j = 1$.

We show the induction step by another induction on the elements in v_e. The base case is clear as e gets inserted as -1.

Now we consider element l in v_e. This is a $-(l-1)$ and was in $l-1$ v's before. Moreover it is left of e_{l-1}. Every $-(l-1)$ that is between those, was in some other v in the same row, or else it would have been taken instead. Thus this $-(l-1)$ is e_l and gets changed into an $-l$. Therefore the first property in question holds.

The second property holds, as once there is no element number l in v_e left, we know that there is no untouched $-(l-1)$ in our part of the path in question left, thus e_l is more to the left.

The third property is more complicated. We point out, that elements, that are number l in v_e with $l > \lfloor r_e/2 \rfloor$, are counted by o. Thus they are the rightmost ones of their μ_m. Due to the Yamanouchi property and Propositions 2.2.3 and 3.3.5 we can argue that in those paths in which they are, there is no other position so far.

Another crucial point for the third property is, that once elements counted by o occur, they also occur in the next row, if there is an element that is larger. The only way how they get less, is when we correct our separation point, thus if a smaller element is considered or there is an empty row.

We now consider elements number $j+1$ up to $|v_e|$ during the insertion process of e.

- An element that is number $j+1$ in v_e is a $-j$ and gets a 0 that is the rightmost 0. This is clear if i is odd. If i is even this follows as then element j needs to be counted as o as well and therefore it is the only element inserted to path j in our area of question. In this case it becomes a 0 on j-level 1.
- The rightmost 0 gets a $-(j+1)$ on level 0 if i is odd just before inserting the next row.
- An element, that is number $j+2$ in v_e, is a 0 before and a j afterwards, if i is odd due to *separate odd*. If i is even, it was and is a $(j-1)$.
- An element that is number $j+m$ in v_e is a $j-m+1$ (respectively $j-m+2$) if i is even (respectively odd).

Now if we insert a c_l into the first path that contains such a $j-m+1$ (respectively $j-m+2$), there are two possible cases. In the first case, c_l is inserted right of the corresponding $-j+m-1$ (respectively $j-m+2$). In this case c_l is larger, and v_c contains all elements our $-j+m-1$ (respectively $j-m+2$) had in its v as well. Thus those are all counted by o. We do not adjust the separation point and the procedure goes on. In the second case, c_l is inserted to the left. Therefore we *adjust the separation point* and the $j-m+1$ (respectively $j-m+2$) becomes a $j+1-m+1$ (respectively $j+1-m+2$). In the same step either a $j-1$ becomes a j or a $j-1$ becomes a 0 (and thus later on a $-(j+1)$) depending on the parity of i. The same happens if for a row there is nothing inserted in the area of question. □

Lemma 3.3.9. *A standard Young tableau Q containing a μ-horizontal step is mapped to a vacillating tableau of cut-away-shape μ by Algorithm 3.*

Proof. Lemma 3.3.8 tells us that if an element is in j different v_e's, it ends up as a $-j$. As elements in μ_j are in exactly j different v_e's (compare with Propositions 2.2.3 and 3.3.5), we get cut away-shape μ. □

Lemma 3.3.10. *If a vacillating tableau has cut-away-shape μ, it is mapped by Algorithm 4 to a standard Young tableau containing a μ-horizontal strip.*

Proof. Let V be a vacillating tableau with cut-away-shape μ. Let Q be its corresponding standard Young tableau and let $\tilde{\mu}$ be the largest partition such that Q contains a $\tilde{\mu}$-horizontal strip. Now by Lemma 3.3.9, V also contains a $\tilde{\mu}$-horizontal strip. If $\tilde{\mu} \supseteq \mu$ we are done. If $\tilde{\mu} \subsetneq \mu$ we show that we get a contradiction.

In this case let p be the largest position in Q that is not in the $\tilde{\mu}$-horizontal strip. We add it to the $\tilde{\mu}$-horizontal strip such that $\tilde{\mu} \subseteq \mu$. Now we know that this does not satisfy one of the three conditions. Therefore we distinguish cases.

1. If the last $\tilde{\mu}_j$ is not a horizontal strip, then p is a descent, which gives a contradiction as Algorithms 3 and 4 are descent preserving and p is not a descent in V.
2. If the word does not satisfy the second condition, the reversed reading word of the according alternative orthogonal Littlewood-Richarson tableau is not Yamanouchi. This gives a contradiction to Propositions 2.2.3 and 3.3.5 and Lemma 3.3.8.
3. If the inequality of the third property is not satisfied there are two possible cases.
    - It could be that a v got longer (this happens exactly if p is in it). For it to be to long, p needs to be at least number $j+1$. However we know, that $p+1$ was inserted at least as often. Therefore p is inserted on level 2. This is a contradiction to being part of a *separation point* due to being number $(j+1)$, compare with Lemma 3.3.8.

- Or it could be that a $\tilde{v}$ in the same row got longer (then p is in this $\tilde{v}$). In this case again there needs to be at least one number $j+1$. The first path with a *separation point* belonging to a position counted by o gets also level 2 positions, which is also a contradiction. □

Thus we have proven (by Lemma 3.3.9 and 3.3.10) the following theorem:

Theorem 3.3.11. *If and only if a standard Young tableau Q contains a μ-horizontal strip, the corresponding vacillating tableau has cut-away-shape μ.*

3.4 Outlook

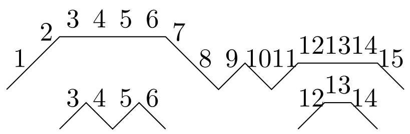
Conjecture 3.4.1. Concatenation of standard Young tableaux, whose row lengths have all the same parity corresponds to concatenation of vacillating tableaux of shape $\emptyset$ in general.

We will prove this for SO(3) in the next chapter. For standard Young tableaux with even row length this was shown in Theorem 3.2.16.

Example 3.4.2. We consider again the standard Young tableaux of Example 1.2.10. ($n = 2k+1 = 5$)

$$\begin{array}{ccccccc} 1 & 2 & 5 & 6 & 9 & 10 & 11 \\ 3 & 4 & 12 \\ 7 & 8 & 13 \\ 14 \\ 15 \end{array} \quad \text{is the concatenation of} \quad \begin{array}{cccc} 1 & 2 & 5 & 6 \\ 3 & 4 \\ 7 & 8 \end{array}, \quad \begin{array}{cc} 1 & 2 \end{array} \quad \text{and} \quad \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}.$$

The corresponding path of the former is the concatenation of the corresponding paths of the latter:



For the other conjecture we need an involution on standard Young tableau called evacuation or Schützenberger involution.

Definition 3.4.3. Let Q be a standard Young tableau with r entries. We define T, the standard Young tableau Q is mapped to via *evacuation*, as follows: T is of the same shape as Q. We determine the filling of T by the following algorithm:

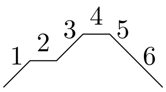
- Let i be the first entry in the first row.
- If i has a lower and/or a right neighbor, we interchange i with the entry to the right or below, depending on which one exists and is smaller.
- We do that until i has neither a lower nor a right neighbor.
- Once this is the case, we delete i and its cell and put $r-i+1$ to the according cell in T.
- We iterate this until every entry of T is determined, thus until 1 is put into the first cell of the first row.

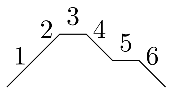
Conjecture 3.4.4. Evacuation (Schützenberger involution) of a standard Young tableau corresponds to the reversal of the corresponding vacillating tableau.

Example 3.4.5. We start with the following standard Young tableau and do evacuation ($n = 2k+1 = 3$):

$$\begin{array}{|c|c|}\hline 1&3\\\hline 2&4\\\hline 5&6\\\hline\end{array}\quad \begin{array}{|c|c|}\hline 2&3\\\hline 4&6\\\hline 5\\\cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 3&6\\\hline 4\\\cline{1-1} 5\\\cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 4&6\\\hline 5\\\cline{1-1}\end{array}\quad \begin{array}{|c|c|}\hline 5&6\\\hline\end{array}\quad \begin{array}{|c|}\hline 6\\\hline\end{array}$$

$$\begin{array}{|c|c|}\hline &\\\hline &\\\hline &\\\hline\end{array}\quad \begin{array}{|c|c|}\hline &\\\hline &\\\hline &6\\\hline\end{array}\quad \begin{array}{|c|c|}\hline &\\\hline &5\\\hline &6\\\hline\end{array}\quad \begin{array}{|c|c|}\hline &\\\hline &5\\\hline 4&6\\\hline\end{array}\quad \begin{array}{|c|c|}\hline &\\\hline 3&5\\\hline 4&6\\\hline\end{array}\quad \begin{array}{|c|c|}\hline &2\\\hline 3&5\\\hline 4&6\\\hline\end{array}\quad \begin{array}{|c|c|}\hline 1&2\\\hline 3&5\\\hline 4&6\\\hline\end{array}$$

The first tableau belongs to 

whereas the second one belongs to 

, which is the reversal of the former one.

Chapter 4

An Alternative Bijection for SO(3)

As mentioned before, in dimension 3 we get stronger results due to the special combinatoric structure. Vacillating tableaux in dimension 3 are Riordan paths (Motzkin paths where horizontal steps are not allowed on the x-axis). We start this chapter by summarizing the special properties that hold in dimension 3. Afterwards, we present an alternative formulation of Bijection B and prove that it is equivalent to the one we had in Chapter 3. We complete this chapter by stating the properties we obtained by this alternative formulation.

The results of this chapter can be found in [3].

4.1 The Special Situation in SO(3)

In dimension 3 not only vacillating tableaux have a special structure, also orthogonal Littlewood-Richardson tableaux are easier. We give a short summery on all the properties that get easier to describe.

As there is only one kind of entry in an alternative orthogonal Littlewood-Richardson tableau in dimension 3 the two properties reduce to one simple property.

Corollary 4.1.1. *For* SO(3) *the set of* aLR_λ^μ *is defines as follows. A tableau* $L \in \mathrm{aLR}_\lambda^\mu$ *is a reverse skew semistandard tableau of inner shape* λ *and* 3 *possibly empty rows. It has* μ_1 *entries* 1*, which form a horizontal strip with at most one entry in the first row.*

Due to this easy form, it is convenient to skip the step of alternative Littlewood-Richardson tableaux and map a pair (L, Q) consisting of an orthogonal Littlewood-Richardson tableau of Kwon $L \in \mathrm{LR}_\lambda^\mu(\mathfrak{d})$ and a standard Young tableau $Q \in \mathrm{SYT}(\lambda)$ directly to a standard Young tableau $\tilde{Q}$. $\tilde{Q}$ consists of Q and a (μ_1)-horizontal strip, thus a horizontal strip with at most one entry in the bottom row. A formulation of Algorithm 1 and Algorithm 2 together with this additional step for SO(3) can be found in the appendix (Algorithm 9 and its reverse Algorithm 10).

Because vacillating tableaux in dimension 3 are Riordan paths, we have just one path to consider in Algorithm 3. Therefore we do not need to consider dependencies between the paths such as *separating points* or *height violations*. A (shorter) formulation of Algorithms 3 and 4 for SO(3) can be found in the appendix (Algorithms 11 and 12).

Throughout this section and in Algorithms 9, 10,11 and 12 we write "level" instead of "1-level", as we have only one path.

4.2 The Algorithm

Now we introduce an alternative formulation of the reversed bijection of Bijection B.

Algorithm 5: Riordan paths to standard Young tableaux: alternative algorithm

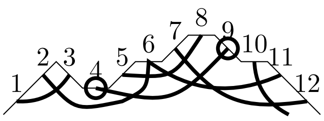
input : vacillating tableau V in terms of a Riordan path
output: standard Young tableau Q
we build a graph, whose vertices are the steps of our Riordan path;
we allow at most one incoming and at most one outgoing edge for each step; moreover, it is not allowed for an edge, that starts (respectively ends) at a step that has an incoming and an outgoing edge, to end (respectively start) at such a step also; we call those edges "double edges";
moreover, some (non double) edges might be marked;
for *entries i in V starting with the leftmost going to right* **do**
- insert (half)edges for entry i between the entries of V as described in Table 4.1;
- connect as many (half)edges as possible with respect to the following rules (any such connecting is admissible);
  - always connect outgoing edges with incoming edges and vice versa, moreover outgoing edges can only be connected to incoming ones to the right;
  - outgoing edges from a -1 cannot be connected with incoming edges unless there are no open double or marked edges;
  - incoming edges of 0's are not allowed to be connected with open double or marked edges;
- put i in the row of Q according to Table 4.1;

return Q;

Theorem 4.2.1. *Algorithm 7 is well-defined and equivalent to Algorithm 4 for vacillating tableaux of shape* $\emptyset$ *and even length.*

Before we prove this we give some examples.

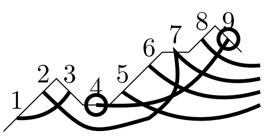
Example 4.2.2. In this example a marked edge occurs and the vacillating tableau has weight $\emptyset$. We point out that while ordinary edges can be open when the path returns to level zero, double or marked edges are never open at such a point.



1	2	4	5	7	10
3	6	8	12		
9	11				

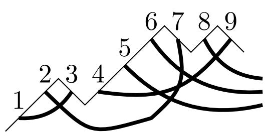
Example 4.2.3. This example shows the crucial point in counting open double or marked edges. Note that before inserting 9, the tableaux, the height, and the next position to consider are the same.

Here we have the situation that the path does not return to zero, thus the shape of the vacillating tableau is not empty. In this example it is (3).



1	2	4	5	6	8
3	7				
9					

whereas



1	2	4	5	6	8
3	7	9			

.

Table 4.1: Conditions for Algorithm 7

<table>
<tr><th>entry</th><th colspan="3">conditions</th><th>kind of edge(s)</th><th>row</th></tr>
<tr><td>1</td><td colspan="3">always (no 0 is active any more)</td><td>outgoing</td><td>1</td></tr>
<tr><td rowspan="6">0</td><td colspan="3"># open double/marked edges $+1 <$ level</td><td>incoming and outgoing</td><td>2</td></tr>
<tr><td rowspan="4"># open double/marked edges $+1 =$ level (this 0 is active, as well as 0's right from here until the next 1 / end of word)</td><td rowspan="2">level and # closed edges have different parity</td><td># active 0's odd</td><td>incoming and outgoing</td><td>2</td></tr>
<tr><td># active 0's even</td><td>incoming</td><td>2</td></tr>
<tr><td rowspan="2">level and # closed edges have same parity</td><td># active 0's odd</td><td>marked outgoing</td><td>1</td></tr>
<tr><td># active 0's even</td><td>outgoing</td><td>1</td></tr>
<tr><td colspan="3"># open double/marked edges $+1 >$ level</td><td>outgoing</td><td>1</td></tr>
<tr><td rowspan="4">-1</td><td colspan="3"># open double/marked edges > 0</td><td>incoming</td><td>3</td></tr>
<tr><td rowspan="3"># open double/marked edges $= 0$</td><td colspan="2">level > 0</td><td>incoming</td><td>2</td></tr>
<tr><td rowspan="2">level $= 0$</td><td># closed edges odd</td><td>incoming</td><td>2</td></tr>
<tr><td># closed edges even</td><td>outgoing</td><td>1</td></tr>
</table>

Proof of Theorem 4.2.1. This proof considers Algorithm 4 and Algorithm 12 for Bijection B. It is more convenient to use the latter. However, using the former and omitting all parts of it that refer to more than one path leads to the same result.

We start this proof with providing properties of Algorithm 7, that prove among others that the algorithm is well-defined:

- Only the first active 0 of a sequence of active 0's can introduce an open double or marked edge or an incoming-only edge.

  To see this we show that once an active 0 occurs the next active 0 in the same sequence will be outgoing-only, as well as once an active 0 has an outgoing-only edge, until the next 1, all active 0's have outgoing-only edges.

  Once there is an outgoing double or marked edge the height condition together with the rule that such edges can only be closed at -1's ensure that for all following active 0's the the third case for the level will be chosen.

  Once there is an incoming-only edge the number of closed edges increases by one. Thus the parity changes. A height decrease also increases the number of closed edges such that we stay in the case of same parity. Moreover, because the number of active 0's is even, which is necessary for the incoming-only case, we stay in the case of outgoing-only edges.

  Once there is an outgoing-only case the number of closed edges only increases at a loss of height. Now this is analogous to the former argument.

- Exactly in those steps where there is an incoming (or of course incoming and outgoing) edge introduced, there is exactly one connection of (half)edges. This is between a former outgoing edge and this new incoming one. In any other case it is not possible to connect (half)edges.

To see this we show that there are always enough outgoing open edges. Every 1 produces an outgoing edge. Every -1 (except for the last one maybe) has an incoming one. Therefore without 0's this is regulated automatically with the level. The only 0 case where the number of outgoing edges is reduced is the one where there is only an incoming edge. However in this case the number of active 0's is even and due to the former point we know that an outgoing-only edge will follow before reaching the bottom level.

- There are never open double or marked edges when the vacillating tableau reaches a bottom point, thus level zero. This holds because there cannot be such new edges when there are as many open ones as the height and those edges need to be closed before an ordinary edge can be closed by a -1.

- The order in which the edges are closed makes no difference. Thus we can assume some first-come-first-served principle to connect (half)edges, and do not consider open (half)edges left of the last 1 to the left on level zero. Moreover, this ensures that the map defined by the algorithm is well-defined.

Next we show that the theorem holds for vacillating tableaux without 0's in 3-row-position. Those are mapped to two-rowed standard Young tableaux in Bijection B.

In this case we have no 0's on a level higher than one and those at level one are always together in even numbers. Therefore we never have any double or marked edges. Thus nothing is put into the third row.

If we go through the path from left to right and find a -1 not on level zero, left of the leftmost 0, if a 0 exists, we will connect this with some 1. If we find a 0 left of the leftmost -1 not on level zero this will connect with the 1 to the left. In both cases it will get to the second row and be our a, exactly as it would be in Bijection B.

As the number of closed edges is now odd the next closed edge will end in a -1 which will also get to the second row and therefore be our b, exactly as it would be in Bijection B.

Now it makes no difference if we extract those or not, because the number of closed edges is even now, there is a new area of active 0's and every other statistic is the same.

Moreover, -1's that belong to the first line are treated correctly as those are at level zero, and will not be considered as new a when the number of closed edges are even.

Next we show that the algorithm finds the 0's in 3-row-position and their associated -1's. It thus follows that the right numbers are put into the third row. We do this inductively.

0's that are at level two or higher when executing Bijection B are recognized automatically. In Bijection B there would be a level decrease of 1 between $\tilde{a}$ (respectively $\tilde{b}$) and a (respectively b). Therefore open double or marked edges increase the needed height. The next -1 not already used by other 0's in 3-row-position is used for a (respectively b).

0's that are on level one, part of a sequence of 0's of odd length, are also recognized. Those are the first positions of active 0's as the level condition shows: $1 = \text{level} - \#$ of open double or marked edges.

Finally we show that it makes no difference for the two-row-case of Algorithm 7 if we do not carry out changes done by extracting the third row.

When extracting a there are three cases:

- $\tilde{a}$ is a non active 0. It becomes a -1 on level one or higher, thus still is on level one or higher.

- $\tilde{a}$ is an active 0. It becomes a -1 on level zero. Whether this is placed in the second or the first row depends in both situations on the parity of closed edges.

- $\tilde{a}$ is a 1. Thus to the right there is an active 0 and the number of active 0's is odd. Again whether this is placed in the second or the first row depends in both situations on the parity of closed edges.

Extracting a changes the parity of closed edges. Thus in the situations where there is a dependency on the parity of closed edges, this change is considered.

When "undoing connect" this happens between two active 0's. While the right one is always put into the first row, there are two possible ways for the left ones (as well as there are for -1's on level zero), depending on the parity of closed edges. Again everything holds due to the change of the parity when extracting a. The same argument holds for "separate".

When extracting b there are two cases that are analogous to the first two cases when extracting a. Note that if $\tilde{a} < \tilde{b} < a < b$ the level and the number of open double or marked edges is reduced by one.

After extracting a and b the parity of open double or marked edges is the same as before as well as the parity of closed edges. Moreover, there is no conflict with active 0's as in an area of active 0's there can be at most one $\tilde{a}$ or $\tilde{b}$. □

Corollary 4.2.4. *Algorithm 8 is an alternative formulation of Algorithm 7. For an entry to insert, it only needs the current entry and four statistics we can calculate using only entries that are already inserted.*

Proof. Algorithm 8 satisfies the mentioned properties due to its formulation.

We see that Algorithm 8 is equivalent to Algorithm 7 when pointing out that:

- a_0 in Algorithm 8 is the number of active 0's in Algorithm 7.
- odm in Algorithm 8 is the number of open double or marked edges from Algorithm 7.
- h in Algorithm 8 is the height before some part of the path. Thus for a 0 or a -1 it is the level and for a 1 it is the level minus one. Although we use the level in Algorithm 7, this does not matter as the neither the height nor the level is considered for a 1 in any of the two algorithms.
- ce is the number of closed edges in Algorithm 7.

The main difference between Algorithm 7 and 8 is that in the latter the decision if an active 0 has an outgoing edge such that this is part of a double edge or a marked edge, is done at the next 1 or the next -1 where the path touches the bottom. However, this makes no difference because for active 0's it holds that for every -1 that is between them, there is another open double or marked edge that can connect with the -1. Besides, only -1's on level zero interrupt a sequence of active 0's. (This explains the $h = 0$ condition for $odm = odm + 1$.)

Moreover, only one active 0's can have a double or marked edge. This holds because once this occurs in the original algorithm, the number of those edges increases and so the last case for 0's occurs until the next 1. □

Remark 4.2.5. One might think that this could be again rewritten in terms of an automaton. However as we need h and odm to be any positive natural number this could not be realized with a finite one. (As we need only the parity of the other statistics those could be implemented as boolean.)

Algorithm 6: Riordan paths to standard Young tableaux: second alternative algorithm

```
input  : vacillating tableau V in terms of a Riordan path
output: standard Young tableau Q
a0 = odm = h = ce = 0;
for entries i in V starting with the leftmost going right do
    switch entry do
        case 1 do
            if a0 odd then odm=odm+1;
            h = h + 1; a0 = 0; put i in the first row of Q;
        case 0 do
            switch odm do
                case < h − 1 do odm = odm + 1; ce = ce + 1; put i in the second row of Q;
                case = h − 1 do
                    a0 = a0 + 1;
                    if h ≢ ce  mod 2 then ce = ce + 1; put i in the second row of Q;
                    else put i in the first row of Q;
                case > h − 1 do a0 = a0 + 1; put i in the first row of Q;
        case −1 do
            h = h − 1;
            if a0 odd and h = 0 then odm = odm + 1; a0 = 0;
            switch odm do
                case > 0 do odm = odm − 1; ce = ce + 1; put i in the third row of Q;
                case = 0 do
                    switch h do
                        case > 0 do ce = ce + 1; put i in the second row of Q;
                        case = 0 do
                            if ce odd then  ce = ce + 1; put i in the second row of Q;
                            else put i in the first row of Q;
return Q;
```

4.3 Properties of Bijection B for SO(3)

Algorithm 7 proves one of the properties for SO(3) that we conjectured for SO$(2k+1)$ in Chapter 3. In particular it proves that the concatenation of two vacillating tableaux of shape $\emptyset$ corresponds to the concatenation of the corresponding standard Young tableaux. We have seen this for general odd dimension and vacillating tableaux of shape $\emptyset$ and even length.

Corollary 4.3.1. *Concatenation of paths with weight* 0 *corresponds to concatenation of standard Young tableaux, all of whose row length have the same parity, when applying Algorithm 7. Therefore this also holds for Bijection B.*

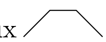
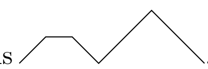
Proof. There cannot be an open double or marked edge when the path reaches the bottom (thus has steps on level zero) due to level conditions. Therefore concatenated parts of paths are independent of each other. □

In particular we get the following theorem, which proves Conjecture 3.4.1 for dimension 3.

Theorem 4.3.2. *For* SO(3) *and Bijection B concatenation of vacillating tableaux of empty shape corresponds to concatenation of standard Young tableaux where all row lengths have the same parity.*

In particular, the following holds:

- *If a vacillating tableau is composed of two concatenated paths with weight* 0*, its corresponding standard Young tableau can be written as concatenation of two standard Young tableaux, such that each of those has row lengths of the same parity.*
- *On the other hand if a standard Young tableau can be written as concatenation of two standard Young tableaux, such that each of those has row lengths of the same parity, its corresponding vacillating tableau is also composed of two concatenated paths with weight* 0.

Example 4.3.3. Concatenating the vacillating tableaux  and  yields .

Whereas concatenating the corresponding standard Young tableaux $\begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline\end{array}$ and $\begin{array}{|c|c|}\hline 1&2\\\hline 3&4\\\hline\end{array}$ yields $\begin{array}{|c|c|c|}\hline 1&4&5\\\hline 2&6&7\\\hline 3\\\cline{1-1}\end{array}$.

Moreover, one might be interested in the pre-images of Dyck paths in Bijection B. We can answer this question also using the concept of concatenation.

Corollary 4.3.4. *Considering Bijection B, Dyck paths, viewed as vacillating tableaux, correspond to those standard Young tableaux that have at most two rows, both of even length, which cannot be written as concatenation of two standard Young tableaux with two rows both of odd length.*

Proof. As inserting the third row produces 0's (compare with the proof of Theorem 3.2.22, *i odd, i even*), the tableaux we are looking for have at most two rows.

First we consider paths of length r with 0's that correspond to two-rowed tableaux. Suppose we insert two numbers left and right of the rightmost such 0 at inserting the third row. Call the label of it c. This can be done by adding $c - 1/2$ and $r + 1/2$ into the third row of the corresponding two-rowed tableau. This would cause *separate*. Thus the path would be composed of two concatenated paths. Thus we can write the new standard Young tableau as concatenation of two tableaux standard Young tableau. As there is only one position in the third row each, both have two odd rows. Thus the original tableau can be written as concatenation of two standard Young tableaux with two rows, both of odd length.

Now we consider a two-rowed standard Young tableau that can be written as the concatenation of two two-rowed standard Young tableaux with rows of odd length. We add an entry in the third row of each. This corresponds to a concatenation of two odd length paths. Thus during the insertion process of this tableau we have *separating*, and for *separating* we need two 0's. Thus the corresponding path for our two-rowed tableau cannot be a Dyck path. □

Remark 4.3.5. Furthermore, due to Corollary 4.2.4 we get:

- For a vacillating tableau of any shape Algorithm 8 produces the right standard Young tableau. For the information on the orthogonal Littlewood-Richardson tableau, we need to add -1's until shape $\emptyset$ is reached as we do in the reversed bijection.
- If one is interested only in the corresponding standard Young tableau Q but not in the orthogonal Littlewood-Richardson tableau, it suffices to run this algorithm on the
- For vacillating tableaux of even length and shape $\emptyset$ we do not need to add $1, 0, -1$ and delete the corresponding entries of the standard Young tableau afterwards.

Corollary 4.3.6. *Algorithm 8 provides an insertion algorithm in the sense of [12, Theorem 6.2 (2)]: if the vacillating tableau* $(v_1, v_2, \ldots, v_{m-1}, v_m)$ *is mapped to the standard Young tableau* Q *of size* m*, the vacillating tableau* $(v_1, v_2, \ldots, v_{m-1})$ *is mapped to the standard Young tableau* $\bar{Q}$ *of size* $m-1$ *which is obtained by deleting the biggest entry* m*.*

Bibliography

[1] Jin Hong and Seok-Jin Kang. *Introduction to quantum groups and crystal bases.* Vol. 42. Graduate Studies in Mathematics. American Mathematical Society, Providence, RI, 2002, pp. xviii+307. ISBN: 0-8218-2874-6. DOI: 10.1090/gsm/042.

[2] Judith Jagenteufel. *A Sundaram type bijection for SO (2k+1): vacillating tableaux and pairs consisting of a standard Young tableau and an orthogonal Littlewood-Richardson tableau.* 2019. arXiv: 1902.03843.

[3] Judith Jagenteufel. "A Sundaram type bijection for SO (3): vacillating tableaux and pairs of standard Young tableaux and orthogonal Littlewood-Richardson tableaux". In: *The Electronic Journal of Combinatorics* 25.3 (2018), p. 3.50. ISSN: 0925-9899.

[4] C. Krattenthaler. "Bijections between oscillating tableaux and (semi)standard tableaux via growth diagrams". In: *J. Combin. Theory Ser. A* 144 (2016), pp. 277–291. ISSN: 0097-3165. DOI: 10.1016/j.jcta.2016.06.015.

[5] Jae-Hoon Kwon. "Combinatorial extension of stable branching rules for classical groups". In: *Trans. Amer. Math. Soc.* 370.9 (2018), pp. 6125–6152. ISSN: 0002-9947. DOI: 10.1090/tran/7104.

[6] M. Lothaire. *Combinatorics on words.* Vol. 17. Encyclopedia of Mathematics and its Applications. A collective work by Dominique Perrin, Jean Berstel, Christian Choffrut, Robert Cori, Dominique Foata, Jean Eric Pin, Guiseppe Pirillo, Christophe Reutenauer, Marcel-P. Schützenberger, Jacques Sakarovitch and Imre Simon, With a foreword by Roger Lyndon, Edited and with a preface by Perrin. Addison-Wesley Publishing Co., Reading, Mass., 1983, pp. xix+238. ISBN: 0-201-13516-7.

[7] Soichi Okada. "Pieri rules for classical groups and equinumeration between generalized oscillating tableaux and semistandard tableaux". In: *Electron. J. Combin.* 23.4 (2016), Paper 4.43, 27. ISSN: 1077-8926.

[8] Robert A. Proctor. "A Schensted algorithm which models tensor representations of the orthogonal group". In: *Canad. J. Math.* 42.1 (1990), pp. 28–49. ISSN: 0008-414X. DOI: 10.4153/CJM-1990-002-1.

[9] Thomas Walton Roby V. *Applications and extensions of Fomin's generalization of the Robinson-Schensted correspondence to differential posets.* Thesis (Ph.D.)–Massachusetts Institute of Technology. ProQuest LLC, Ann Arbor, MI, 1991, (no paging).

[10] Martin Rubey, Bruce E. Sagan, and Bruce W. Westbury. "Descent sets for symplectic groups". In: *J. Algebraic Combin.* 40.1 (2014), pp. 187–208. ISSN: 0925-9899. DOI: 10.1007/s10801-013-0483-4.

[11] Richard P. Stanley. *Enumerative combinatorics. Vol. 2*. Vol. 62. Cambridge Studies in Advanced Mathematics. Cambridge University Press, Cambridge, 1999, pp. xii+581. ISBN: 0-521-56069-1; 0-521-78987-7. DOI: 10.1017/CBO9780511609589.

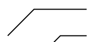
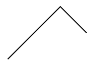
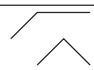
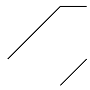
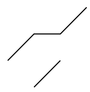
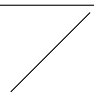
[12] John R. Stembridge. "Rational tableaux and the tensor algebra of gl_n". In: *J. Combin. Theory Ser. A* 46.1 (1987), pp. 79–120. ISSN: 0097-3165. DOI: 10.1016/0097-3165(87)90077-X.

[13] Sheila Sundaram. *ON THE COMBINATORICS OF REPRESENTATIONS OF THE SYMPLECTIC GROUP*. Thesis (Ph.D.)–Massachusetts Institute of Technology. ProQuest LLC, Ann Arbor, MI, 1986, (no paging).

[14] Sheila Sundaram. "Orthogonal tableaux and an insertion algorithm for $\mathrm{SO}(2n+1)$". In: *J. Combin. Theory Ser. A* 53.2 (1990), pp. 239–256. ISSN: 0097-3165. DOI: 10.1016/0097-3165(90)90059-6.

Appendix

Table 2: List of all tableaux with $n = 5$, $r = 3$. Note that there is not necessarily an orthogonal Littlewood-Richardson tableau for every combination of μ and λ with $\mu \subseteq \lambda$.

λ	μ	L	$\tilde{L}$	Q	V
$(1,1,1)$	$(1,1)$	[1] [1] [1]	[ ] [ ] [ ] [2] [1]	[1] [2] [3]	
$(2,1)$	(1)	[1] [1|2] (column 1 over 2)	[ ][ ] [ ][1]	[1][2] [3]	
				[1][3] [2]	
	$(2,1)$	[1|2] (column 1 over 2) [1]	[ ][ ] [ ][2] [1][1]	[1][2] [3]	
				[1][3] [2]	
(3)	(1)	[3] [1|2] (column 1 over 2)	[ ][ ][ ][1]	[1][2][3]	
	(3)	[1] [2] [3]	[ ][ ][ ][1] [1][1]	[1][2][3]	

Algorithms

Algorithm 7: Algorithm to Bijection A for SO(3)

input : orthogonal Littlewood-Richardson tableau $L \in \mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})$, standard Young tableau $Q \in \mathrm{SYT}(\lambda)$

output: standard Young tableau $\tilde{Q}$ with 3 (possibly empty) rows, all of even or all of odd length, contains the original standard Young tableau and μ_1 additional positions that form a horizontal strip with at most one element in the first row, that is filled increasingly from left to right
the partition μ, unmodified

rotate L by $\frac{\pi}{2}$ and shift cells such that it becomes the row-strict tableau $\tilde{L}$, row i being filled with $1, 2, \ldots, \lambda_i$;

add cell labeled x to $\tilde{L}$ in row $i+1$ for each position in row i that was not in column i from the right;

add cells labeled x to $\tilde{L}$ in row 3 such that the number of cells labeled x is μ_1;

if *not all rows of* $\tilde{L}$ *have the same parity* **then** `/* adjust parity */`
- **if** *total number of cells is odd* **then**
  - move last cell of the bottommost even row to the other even row;
- **if** *total number of cells is even* **then**
  - move last cell of the bottommost odd row to the other odd row;

add numbers $|\lambda|+1, |\lambda|+2, \ldots$ increasingly from left to right to Q at the positions of the cells in $\tilde{L}$ labeled x to obtain $\tilde{Q}$;

return $(\tilde{Q}, \mu)$;

Algorithm 8: Reversed Algorithm to Bijection A for SO(3)

---

input : standard Young tableau $\tilde{Q} \in \mathrm{SYT}(\tilde{\lambda})$ with 3 rows, all of even or all of odd length, a partition $\mu = (\mu_1)$, such that the skew-diagram containing the μ_1 largest entries in $\tilde{Q}$ form a horizontal strip with at most one position in the first row, filled from left to right increasingly

output: orthogonal Littlewood-Richardson tableau $L \in \mathrm{LR}^{\mu}_{\lambda}(\mathfrak{d})$, standard Young tableau $Q \in \mathrm{SYT}(\lambda)$

let L be the row-strict tableau of the same shape as $\tilde{Q}$, with row i filled with $1, 2, \ldots, \tilde{\lambda}_i$;
delete the cells in $\tilde{Q}$ with the μ_1 largest entries to obtain Q;
replace the entries in the corresponding cells in L with x's;
rotate L by $-\frac{\pi}{2}$;
for *each column of L* **do** `/* mark elements for tail */`
- **if** *there is an odd number of cells labeled x in this column* **then**
  - mark the bottommost cell in this column that is not marked yet and contains a number, if there exists none, mark the bottommost unmarked cell in the column to the right;
  - delete the bottommost cell labeled x;
- **if** *there are still cells labeled x in this column* **then**
  - beginning at the bottom, mark as many cells, that are not marked yet and contain numbers, in the column to the right, as there are cells labeled x in the current column;
  - delete the cells labeled x in the current column;

shift all cells that are marked to the leftmost column;
separate the rightmost column from the other ones such that we obtain two tableaux;
shift the leftmost column down such that the tail has length μ_1;
if *the second column has odd length and $\mu_1 \neq 0$* **then** `/* fix parity */`
- find the smallest entry in the two-column tableau, that can change columns, such that the resulting tableau, after shifting columns to obtain a tail of length μ_1, is an orthogonal Littlewood-Richardson tableau; move this entry to the other column in the two-column tableau and shift the columns such that the tail has length μ_1.

return *(L, Q)*

Algorithm 9: Algorithm to Bijection B for SO(3)

```
input  : standard Young tableau Q with at most 3 rows, all of them of even length
output: vacillating tableau V of shape ∅ of even length
/* Inserting 1st row                                                              */
construct word V = (1, −1, 1, −1, ..., 1, −1) with same length as the first row of Q;
label the letters with the numbers of the first row of Q;
/* Inserting 2nd row                                                              */
for pairs a, b in the second row of Q, starting with the rightmost pair, going left do
    insert −1 at b;                                                  /* insert b */
    if position to the right of a is −1 not labeled with b then  /* insert a case 1 */
        insert 0 at a; change −1 to the right of a into 0;
    else                                                          /* insert a case 2 */
        insert −1 at a; change next −1 to the left of a into 1;
    change pairs of 1, −1 between a and b into 0, 0;
/* Inserting 3rd row                                                              */
for pairs a, b in the third row of Q, starting with the rightmost pair, going left do
    insert −1 at b; change next −1 to the left of b into 0, let b̃ be its label; /* insert b */
    start at b and let c be the current position, let C be an empty list;
    while ã undefined do
        if c is 1 on level 0 then                                       /* connect */
            change c and the position to the left into 0, put their labels into C;
        else if the number of positions strictly to the left is even, c and the position to the
         left are 0's on level 1 and a is not inserted then              /* separate */
            change c into 1 and the position to the left into −1;
        if a is directly to the left of c then                         /* insert a */
            insert −1 at a;
        else if a is inserted then
            if c is −1 not labeled with a then                           /* case 1 */
                change c into 0, let ã be its label;
            else if c is 0, not in C, left of b̃ then                     /* case 2 */
                change c into 1, let ã be its label;
        go one position to the left;
forget the labeling of V and return V;
```

Algorithm 10: Reversed Algorithm to Bijection B for SO(3)

input : vacillating tableau V of shape $\emptyset$ of even length r
output: standard Young tableau Q with at most 3 rows, all of them of even length

attach labels $\{1, 2, \ldots, r\}$ to V;
`/* Extracting 3rd row */`
while *there are* 0*'s in 3-row-position* **do**
 let p be the leftmost 0 in 3-row-position and p' be the position to its right;
 if p *is on level* 1 *and* p' *is a* 1 **then** `/* extract a case 2 */`
  change p' into 0, let $\tilde{a}$ be p';
 else change p into -1, let $\tilde{a}$ be p; `/* extract a case 1 */`
 let a be the next -1 to the right be a, delete a;
 start at $\tilde{a}$, let the current position be c;
 while $\tilde{b}$ *undefined* **do**
  if c *is* -1 *on level* 0 **then** `/* undo separate */`
   change c and the position to the right into $0, 0$;
  else if *the number of positions strictly to the left is odd, both* c *and the position to the right are* 0 *on level* 1 **then** `/* undo connect */`
   change c and the position to the right into $-1, 1$;
  if c *is* 0 *in 3-row-position* **then** `/* extract b */`
   change c into -1, let $\tilde{b}$ be c; let b be the next -1 to the right, delete b;
  go one position to the right;
 insert a, b into the third row of Q;
`/* Extracting 2nd row */`
while $V \neq (1, -1, 1, -1, \ldots, 1, -1)$ **do**
 let a be the leftmost position that is neither a 1 nor on level 0; `/* extract a */`
 if a *is* -1 **then** change the leftmost 1 on level 1 into -1; `/* case 2 */`
 else change 0 to the right of a into -1; `/* case 1 */`
 delete a; let b be the next -1 to the right of the new -1, delete b; `/* extract b */`
 change $0, 0$ on level 0 into $1, -1$;
 insert a, b into the second row of Q;
`/* Extracting 1st row */`
insert labels still in the word into the first row of Q;
return Q;

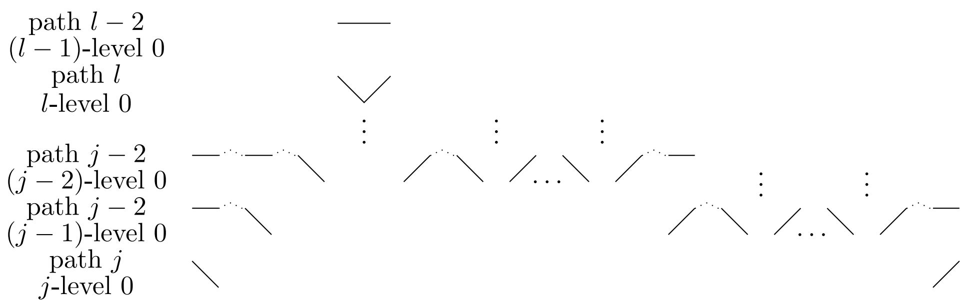
A Guide to understand Algoritm 3

Main Ideas

- We insert new elements with ╲ and change the next ╲ into ‾. We insert those into the next path. Each two rows a new path is inserted with ╱╲╱╲ ⋯ ╱╲ where the newest path so far had ‾ steps. There are extra rules for inserting something into a path the first time. We always go through the paths from right to left.
- We want tableaux that consist of all-odd-length-rows to be mapped to odd-length paths. Therefore, if we are between a and b at certain points we mark and sometimes change certain structures. We call them "separation points".
- If an inserted step causes path l to be higher than path $l-1$, the rightmost position where this happens is a ╱ step in path $l-1$. We change it into ‾ in path $l-1$ and insert a ╱ step into path l. Moreover we act as if the original insertion has not happened. This procedure can also cause path $l-1$ to be higher than path $l-2$. We deal with this the same way and act as if the last insertion into path $l-1$ has not happened too.

Main rules

- We insert one row after the next from top to bottom, from left to right in pairs a, b. i counts the row and $j = \lfloor i/2 \rfloor$ denotes which path is the newest.
- The first row is inserted as ╱╲╱╲ ⋯ ╱╲ into path 1.
- For each a, b we set $a_1 = a$, $b_1 = b$, $a_l = b_l = 0$ for $l = 2, 3, \dots, j+1$. and go through the paths from right to left. We insert a_1 and b_1 with ╲ into path 1.
- a_{l+1} (respectively b_{l+1}) can be defined only if a_l (respectively b_l) is defined.
- a_l can be defined only if b_l is defined.
- Positions we mark are always $\pm l$ on l-level 0 or 0's close to such positions (however we never ask for a 0 step whether it is marked). We mark at certain steps while considering path j. At such a point we mark for all l all $\pm l$ on l-level 0 between those steps, and in certain cases we change those. For a better imagination we give the following sketch. A separation point looks like this after inserting an odd row. (After inserting an even row the two positions in path j are horizontal steps.)



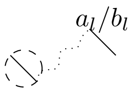
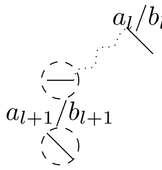
- A marked position is not allowed to be defined as b_l.
- a_l directly between two marked positions get marked too.

- Defined a_l (respectively b_l) are allways right of the current position. (There is one exeption, if a_{j+1} is  in path j and i is even.)

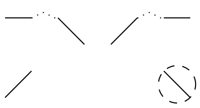
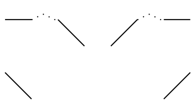
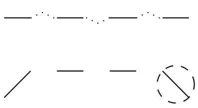
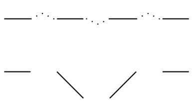
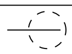
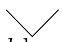
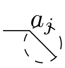
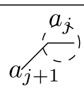
We now explain the steps of Algorithm 3. The current position is circled. If an l occurs, it means "if there is an l such that".

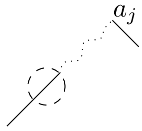
Procedure	Initial situation	Goal

Insertion of a pair a, b

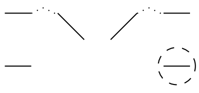
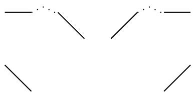
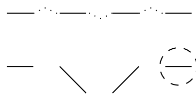
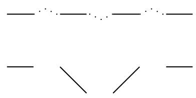
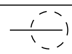
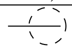
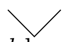
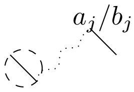
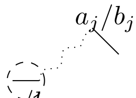
$/ * a_{l+1} * /$ or $/ * b_{l+1} * /$	 path l b_{l+1}/a_{l+1} not defined	 path l path $l+1$

For i **even:**

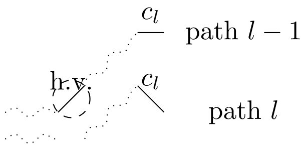
/*adjust separation point*/	 path $j-1$ $(j-1)$-level 0 path j j-level 0 b_j not defined (levels before b inserted) and between the two steps in path j  path $l-1$ $(l-1)$-level 0 path l l-level 0	 path $j-1$ $(j-1)$-level 0 path j j-level 0 and between the two steps in path j  path $l-1$ $(l-1)$-level 0 path l l-level 0 + mark all changed positions
/*mark it + connect */	 path j j-level 0 (before b_j was inserted) between b_j and a_j (which is not defined)	 path j + mark positions for all $l < j$  path l l-level 0 (before b_l was inserted) between those two steps
$/ * a_{j+1}\ 1 * /$	 path j a_{j+1} not defined	 path j

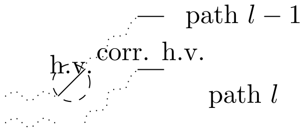
$/* a_{j+1}\ 2 */$	 a_j path l j-level 0 a_{j+1} not defined	 a_j path l j-level 0
$/* b_{j+1} */$	b_{j+1} not defined current position directly left of b_j	b_{j+1} is current position

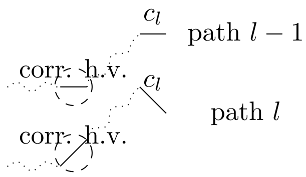
For i odd:

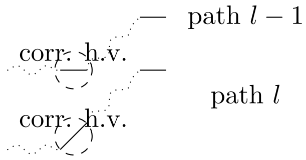
/∗adjust separation point∗/	 path $j-1$ $(j-1)$-level 0 path j j-level 0 b_j not defined (levels before b inserted), there are even many steps strictly left of the current position and between the two steps in path j  path $l-1$ $(l-1)$-level 0 path l l-level 0	 path $j-1$ $(j-1)$-level 0 path j j-level 0 and between the two steps in path j  path $l-1$ $(l-1)$-level 0 path l l-level 0 + mark all changed positions
/∗connect ∗/	 path j j-level 0 (before a_j was inserted) between b_j and a_j (which is not defined)	 path j
/∗mark it + separate ∗/	 path j j-level 0 (before a_j was inserted) between b_j and a_j (which is not defined) there are even many steps strictly left of the current position	 path j + mark those and the following positions for all $l < j$  path l l-level 0 (before b_l was inserted) between those two steps
$/* a_{j+1} */$ or $/* b_{j+1} */$	 a_j/b_j path j b_{j+1}/a_{j+1} not defined	 a_j/b_j path j a_{j+1}/b_{j+1}

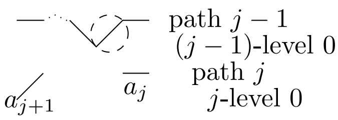
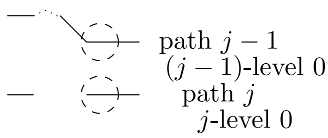
Height Violations:

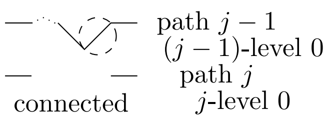
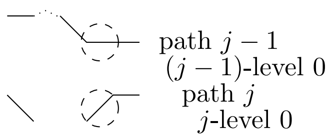
<table>
<tr>
<td>/*height violation*/</td>
<td>

or


h.v. ... height violation
corr. ... corrected</td>
<td>

+ c_l is undefined again
or


+ c_l is undefined again</td>
</tr>
<tr>
<td>/*special case i even */</td>
<td>
</td>
<td>

a_j, a_{j+1} undefined again</td>
</tr>
<tr>
<td>/*special case i odd */</td>
<td>
</td>
<td>

b_j, b_{j+1} undefined again</td>
</tr>
</table>

Curriculum Vitae

Personal Information

Name	Judith Jagenteufel, née. Braunsteiner
Date of Birth	August 31$^{\text{th}}$, 1991
Place of Birth	Stockerau, Austria

Education

since 11/2015	Doctoral Programme in Technical Mathematics (TU Wien)
10/2013-10/2015	Master Programme in Technical Mathematics (TU Wien)
10/2009-10/2013	Bachelor Programme in Technical Mathematics (TU Wien)

PHD Thesis

Title	A Sundaram type bijection for $\mathrm{SO}(2k+1)$: vacillating tableaux and pairs consisting of a standard Young tableau and an orthogonal Littlewood-Richardson tableau
Advisor	Privatdoz. Dr.rer.nat. Martin Rubey

Master Thesis

Title	Combinatorial r-Species and their Substitution
Advisor	Privatdoz. Dr.rer.nat. Martin Rubey

Professional Experience

since 03/2016	Project Assistant (FWF) Institute of Discrete Mathematics and Geometry, TU Wien
10/2013-02/2016	Teaching Assistant Institute of Discrete Mathematics and Geometry, TU Wien
10/2011-02/2013	Tutor Institute of Discrete Mathematics and Geometry, TU Wien

Publications

A Sundaram type bijection for $\mathrm{SO}(3)$*: vacillating tableaux and pairs of standard Young tableaux and orthogonal Littlewood-Richardson tableaux*, in The Electronic Journal of Combinatorics 25.3 (2018) p. 3.50. issn: 0925-9899

A Sundaram type bijection for $\mathrm{SO}(2k+1)$*: vacillating tableaux and pairs consisting of a standard Young tableau and an orthogonal Littlewood-Richardson tableau*, extended abstract, accepted for publication in the proceedings volume of Séminaire Lotharingien Combinatoire of the 31th International Conference on "Formal Power Series and Algebraic Combinatorics", July 1–5, 2019, University of Ljubljana, Slovenia

Preprints

A Sundaram type bijection for $\mathrm{SO}(2k+1)$*: vacillating tableaux and pairs consisting of a standard Young tableau and an orthogonal Littlewood-Richardson tableau*, preprint 2019, submitted