A direct bijective proof of the hook-length formula.

Jean-Christophe Novelli[†]
Igor Pak[‡]
Alexander V. Stoyanovskii[§]

This paper presents a new proof of the hook-length formula, which computes the number of standard Young tableaux of a given shape. After recalling the basic definitions, we present two inverse algorithms giving the desired bijection. The next part of the paper presents the proof of the bijectivity of our construction. We finish with some examples.

1 Introduction

The aim of this paper is to give a bijective proof of the hook-length formula for the enumeration of standard Young tableaux of a given shape. This formula was discovered by Frame, Robinson and Thrall in 1954 [1] and since then it has been the object of much study. Many proofs have been published based on different approaches, but none of them is considered satisfactory. We refer to the paper [9] for a well written review of the different proofs and their history, but we want to recall a few of them and some related papers which had a strong impact on us (see also the references in [5, 10]).
First, we should mention the remarkable paper of Schützenberger [11]. It is not directly related to the hook-length formula but contains the famous *jeu de taquin* algorithm. Our bijection is based on this procedure.
The first major steps in the understanding of the hook-length formula were made by Hillman and Grassl [4] and Stanley [12]. Their two beautiful bijections combined give the result, although an algebraic step is also needed.

[†]Université Paris 7 – LIAFA – 2, place Jussieu - 75251 Paris Cedex 05 – email: jcn@litp.ibp.fr
[‡]Department of Mathematics – Harvard University – Cambridge - MA 02138. email: Pak@math.harvard.edu.
[§]Department of Mathematics – Independent University – Moscow - Russia. email: stoyan@sch57.mc.msk.su.

1365–8050

One of the most amazing proofs of the hook-length formula was found by Greene, Nijenhuis and Wilf [3]. Their probabilistic approach leads to a bijection as well [13], but some of the details are complicated. A direct bijective proof was first found by Franzblau and Zeilberger in their celebrated paper [2]. Their construction is very nice and in some way similar to ours, but in this case it is the proof that is a bit complicated.

We also want to point out the paper of Krattenthaler [6], which contains an involutive proof of a q-analogue of the hook-length formula and includes an extension to the shifted case.

The bijection we present in this paper was announced by the last two authors a few years ago in [8], but this is the first complete version of the proof. All the results of this paper have been generalized to trees and shifted tableaux, which are the other two known cases where hook-length formulas exist. These results will appear elsewhere.

Our paper is organized as follows. In Section 2, we recall the standard definitions and notation. In Section 3, we state the result and explain how it can be proved bijectively. In Sections 4 and 5, we define the algorithm and its inverse. Section 6 is devoted to proving that these algorithms indeed define a bijection. Section 7 contains several examples that will help the reader to understand how both algorithms work.

Acknowledgments. The authors are grateful to Bruce Sagan for reading and correcting the preliminary version of the paper. The authors also wish to thank Sergey Fomin, Alexandre Kirillov and Christian Krattenthaler for helpful remarks. The first author wishes to thank Jean-Yves Thibon for all his helps and comments. The second author wishes to thank the Fanny and John Hertz Foundation for their partial support.

2 Definitions and notation

Let us recall the following standard definitions (see, e.g., [7, 5]). A *partition* of an integer n is a nonincreasing sequence of nonnegative integers $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_p)$ such that $|\lambda| = \lambda_1 + \lambda_2 + \cdots + \lambda_p = n$. A Ferrers diagram is a graphic representation of a partition λ. For example, the partition $(5, 2, 2, 1)$ is represented as follows.

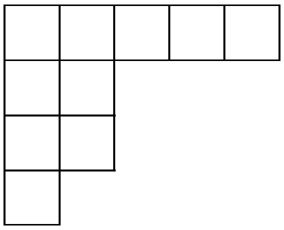

Figure 1: A Ferrers diagram.

The first row contains λ_1 cells, the second row λ_2 cells and so on. To take a more formal definition of these diagrams, let us consider the set of pairs $(i, j) \in \mathbb{Z}^2$ satisfying the conditions $1 \leq j \leq \lambda_i$ where $(\lambda_1, \lambda_2, \ldots, \lambda_p)$ is a partition. The Ferrers diagram associated with this partition is then the set of 1×1 squares with centers at the points (i, j) where the pair (i, j) runs over the set defined previously. In the rest of this paper, we will identify the cells of a diagram with their coordinates when no confusion is possible. A *Young tableau* is a Ferrers diagram filled bijectively with the integers $1, 2, \cdots, |\lambda|$. A tableau is said to be ordered up to position (i_0, j_0) if the numbers increase from top to bottom along the columns and from left to right along the rows for all (i, j) such that either $j > j_0$, or $j = j_0$ and $i \geq i_0$. A *standard tableau* is a tableau ordered up to position $(1,1)$ (see Figure 2).

5	3	7
2	5	
6	1	

1	3	7
2	4	
5	6	

Figure 2: A tableau and a standard tableau.

The *hook* of cell (i,j) of a Ferrers diagram is the set of cells that are either in row i weakly right of (i,j), or in column j weakly below (i,j) (see Figure 3). Let $h(i,j)$ denote the cardinality of this set.

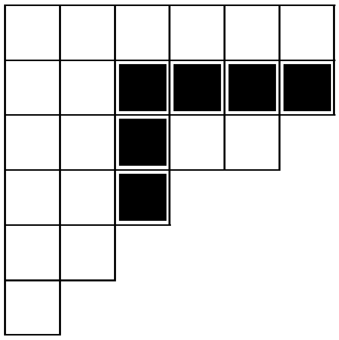

Figure 3: The hook of the cell $(2,3)$.

The conjugate of a partition λ (defined as usual, see [7], for example) is denoted by λ'. In the sequel, we will consider Young tableaux of a fixed shape λ with $|\lambda| = n$.

In our bijection, we will use a new object called a *hook function*. Geometrically, it corresponds to filling a Ferrers diagram with integers satisfying some conditions. More precisely, each cell of the diagram is filled with an integer between minus the number of cells strictly below in its column and plus the number of cells strictly right of it in its row (see Figure 4). More formally, we define the hook function as a mapping f from λ to $\mathbb{Z}$ that satisfies the condition

$$\forall (i,j) \in \lambda,\ -(\lambda'_j - i) \leq f(i,j) \leq (\lambda_i - j).$$

For example, the value of the hook function on the cell $(2,3)$ in Figure 3 must be between -2 and $+3$.

-1	-1	-2
2	-2	0
1	1	0
0	0	

Figure 4: A hook function.

3 The hook-length formula

Our goal is to prove the next well-known theorem bijectively.

Theorem 3.1

$$f_\lambda = \frac{n!}{\prod\limits_{(i,j)\in\lambda} h(i,j)},$$

where f_λ is the number of standard tableaux of shape λ.

The bijection we are going to construct is between the following sets:

- I: the pairs of the form:
  - A standard tableau of shape λ,
  - A hook function.
- II: the tableaux of shape λ.

It is clear that the cardinality of II is $n!$ and that the cardinality of I is

$$f_\lambda \times \prod_{(i,j)\in\lambda} (\lambda_i + \lambda'_j - i - j + 1) = f_\lambda \times \prod_{(i,j)\in\lambda} h(i,j).$$

The bijection consists of two algorithms that are inverse to each other.

4 Algorithm $II \mapsto I$.

To transform an element of II into an element of I, we use two simple algorithms that will be useful in the proofs and will be the building blocks of the main algorithm. The first one is Algorithm P which is just backward 'jeu de taquin' as described by Schützenberger in [11] (using the terminology of B. Sagan, see [10]).

Algorithm P.
`INPUT:` A pair (A, a) consisting of a tableau A and an integer $a \leq n$.
`OUTPUT:` A tableau.

- Step 0. Denote by (i_0, j_0) the coordinates of the cell of A that contains a.
- Step 1. Put

$$b = A(i_0, j_0+1) \text{ if } j_0 < \lambda_{i_0}, \text{ and } b = n+1 \text{ otherwise},$$
$$c = A(i_0+1, j_0) \text{ if } i_0 < \lambda'_{j_0}, \text{ and } c = n+1 \text{ otherwise}.$$

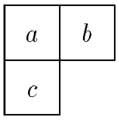

Figure 5: Generic positions of b and c.

- Step 2.
  - If a is greater than b or c then exchange a with the smaller of b and c. This defines a new tableau B. Go back to Step 0, with $A = B$.
  - If a is smaller than b and c, the algorithm finishes and outputs A.

Example 4.1 We give in Figure 6 the sequence of tableaux obtained by applying Algorithm P to the first tableau and the integer 5. The integers corresponding to b and c are underlined in each tableau (when they belong to it).

$$\begin{array}{|c|c|}\hline \mathbf{5} & \underline{1} \\ \hline \underline{2} & 3 \\ \hline 4 & \multicolumn{1}{c}{} \\ \cline{1-1} \end{array} \longrightarrow \begin{array}{|c|c|}\hline 1 & \mathbf{5} \\ \hline 2 & \underline{3} \\ \hline 4 & \multicolumn{1}{c}{} \\ \cline{1-1} \end{array} \longrightarrow \begin{array}{|c|c|}\hline 1 & 3 \\ \hline 2 & \mathbf{5} \\ \hline 4 & \multicolumn{1}{c}{} \\ \cline{1-1} \end{array}$$

Figure 6: An example of an application of Algorithm P.

Definition 4.2 *Totally order the cells of a tableau by reverse lexicographic order on their coordinates. In other words, we order the cells starting from the right-most column moving to the left, and from bottom to top inside the columns. Figure 7 shows the numbering of the cells of a tableau corresponding to this order.*

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

Figure 7: The order of the cells.

We will use the *successor* map s which sends each cell (except the last) to the next one in this order. We will also use the *predecessor* map, denoted s^{-1}.

Algorithm 1.
`INPUT:` `A triple` $(A, f, (i_0, j_0))$ `consisting of a tableau` A`, a hook function` f`, and a cell` $(i_0, j_0) \neq (1, 1)$ `such that the tableau is ordered up to` (i_0, j_0)`.`
`OUTPUT:` `A triple` $(B, g, s(i_0, j_0))$ `consisting of a tableau` B`, a hook function` g `and the successor` $s(i_0, j_0)$ `of the coordinates of the input.`

- Step 0. Set $(i_1, j_1) = s(i_0, j_0)$ and $a = A(i_1, j_1)$.
- Step 1. Compute B by applying Algorithm P to (A, a).
- Step 2. Let (i_2, j_2) be the position of a in B.
  - For all $i_1 \le i < i_2$, put $g(i, j_1) = f(i+1, j_1) - 1$,
  - $g(i_2, j_1) = j_2 - j_1$,
  - $g(i, j) = f(i, j)$ otherwise.

The output is $\big(B, g, (i_1, j_1)\big)$.

Note 4.3 By standard properties of jeu de taquin, tableau B is ordered up to position (i_1, j_1). Moreover, it follows from Step 2 that g is a hook function. This establishes that Algorithm 2 is well-defined.

We can now associate with every element of II an element of I.

Algorithm 2**.**
`INPUT: An element` A `of` II.
`OUTPUT: An element` (A, f) `of` I.

- Step 0. Set $f = 0$ and (i_0, j_0) the coordinates of the smallest cell of A in our total order.
- Step 1. Iterate Algorithm 1 on $\big(A, f, (i_0, j_0)\big)$ until $(i_0, j_0) = (1, 1)$. The algorithm then finishes and outputs (A, f).

Note 4.3 shows, in particular, that Algorithm 2 gives a standard tableau as output.

5 Algorithm $I \mapsto II$

To transform an element of I into an element of II, we build the main algorithm from two others as in Section 4. The first one, Algorithm P', is forward 'jeu de taquin.'

Algorithm P'**.**
`INPUT: A triple` $\big(T, a', (i'_0, j'_0)\big)$ `consisting of a tableau` T `and an integer` $a' \le n$.
`OUTPUT: A tableau.`

- Step 0. Denote by (i'_1, j'_1) the position of a' in T. If $(i'_1, j'_1) \ge (i'_0, j'_0)$ then the algorithm finishes and outputs T.
- Step 1. Put

$$b' = T(i'_1, j'_1 - 1) \text{ if } j'_1 > j'_0, \text{ and } b' = 0 \text{ otherwise,}$$

$$c' = T(i'_1 - 1, j'_1) \text{ if } i'_1 > 0, \text{ and } c' = 0 \text{ otherwise.}$$

$*$	c'
b'	a'

Figure 8: Generic positions of b' and c'.

- Step 2. Exchange a' with the greatest of b' and c'. This defines a new tableau U. Go back to Step 0 with $T = U$.

The previous algorithm finishes after a finite number of steps since a' will stop at some cell $\leq (1, j'_0)$ by the rules defined above. Note also the difference between Algorithm P and Algorithm P'. In Algorithm P', we do not compare a' with b' or c' to decide whether it can move or not. When a' stops moving depends on its current position and is independent from its value.

Definition 5.1 *Let us consider the path of the integer a' when applying Algorithm P'. At each step of Algorithm P', we code its move by N if it exchanges with c' (a north move) and by W if it exchanges with b' (a west move). We obtain a word by concatenating all the letters W and N. The* coding *of the path of a' is this word read from right to left.*

Example 5.2 In Figure 9 we give the sequence of tableaux obtained by applying Algorithm P' to the first tableau, the number 8 and the cell $(1, 2)$. The integers corresponding to b' and c' are underlined in each tableau (when they belong to it). The word obtained is NNW so that the coding of the path of 8 is WNN.

2	1	6
4	3	$\underline{7}$
9	$\underline{5}$	**8**

$\longrightarrow$

2	1	$\underline{6}$
4	$\underline{3}$	**8**
9	5	7

$\longrightarrow$

2	$\underline{1}$	**8**
4	3	6
9	5	7

$\longrightarrow$

2	**8**	1
4	3	6
9	5	7

Figure 9: An example of an application of Algorithm P'.

We will order the alphabet $\{N, W, \emptyset\}$ by setting $N < \emptyset < W$ and consider the associated lexicographic order.

Definition 5.3 *Let U be a tableau, g' a hook function and (i'_1, j'_1) the coordinates of a cell of U. The set of* candidate *cells associated with $\big(U, g', (i'_1, j'_1)\big)$ is the set of all cells indexed by (i, j) such that $i \geq i'_1$, $g'(i, j'_1) \geq 0$ and $j = j'_1 + g'(i, j'_1)$. The set of candidate elements, or simply candidates, is the set of the integers that are in candidate cells. The maximum candidate (element or cell) is the one with the lexicographically largest coding.*

By construction, there is at most one candidate cell in each row. Notice that there always exists a candidate element: the hook function maps the bottom cell of any column to a nonnegative integer.

Note 5.4 Geometrically, we can define the maximum candidate as the one with the top-most and right-most path. If one path is contained in another, the choice depends on the move done by the longer one just before joining the shorter one. We will see further that paths cannot cross and that this property makes everything well-defined (see Section 6.2).

Algorithm $1'$.
`INPUT:` A triple $(T, g', (i'_0, j'_0))$ consisting of a tableau T, a hook function g', and a cell (i'_0, j'_0) (other than the least one) such that the tableau is ordered up to (i'_0, j'_0) .
`OUTPUT:` A triple $(U, f', s^{-1}(i'_0, j'_0))$ consisting of a tableau U, a hook function f', and the predecessor $s^{-1}(i'_0, j'_0)$ of the cell of the input.

- Step 0. Find the maximum candidate element p and denote its cell in T by (i'_1, j'_1).
- Step 1. Let U be the output of Algorithm P' applied to $(T, p, (i'_0, j'_0))$.
- Step 2.
  - For all $i'_0 < i \leq i'_1$, put $f'(i, j'_0) = g'(i-1, j'_0) + 1$,
  - $f'(i'_0, j'_0) = 0$,
  - $f'(i, j) = g'(i, j)$ otherwise.

The output is $(U, f', s^{-1}(i'_0, j'_0))$.

Note 5.5 Notice that U is not *always* ordered up to $s^{-1}(i'_0, j'_0)$. In fact, if the maximum candidate moves to position (i'_0, j'_0) in Step 1 then U is ordered up to $s^{-1}(i'_0, j'_0)$. But if it moves to another cell this may not be the case. Notice also that it is not clear that Algorithm $2'$ is well-defined: we need to prove that f' is a hook function, which is not obvious, and that at each step the maximum candidate moves to position (i'_0, j'_0). Both properties will be established in Section 6.3.

We can now associate with every element of I an element of II.

Algorithm $2'$.
`INPUT:` An element (T, g') of I.
`OUTPUT:` An element T of II.

- Step 0. Set $(i'_0, j'_0) = (1, 1)$.
- Step 1. Iterate Algorithm $1'$ on $(T, g', (i'_0, j'_0))$ until (i'_0, j'_0) is the least cell of T. The algorithm then finishes and outputs T.

6 Proofs

To prove that we have a bijection, we need to show that Algorithms 2 and $2'$ are inverses. Since these procedures are successive applications of either Algorithm 1 or Algorithm $1'$, we only have to prove that these algorithms are inverses. Unfortunately, this is not true in general: given a triple that satisfies the conditions of Algorithm 1, and applying successively Algorithm 1 and Algorithm $1'$, we do not necessarily obtain the input triple. But we can find a subset of all triples for which this is true. Since this subset contains all the intermediate triples (when applying Algorithm 2 or Algorithm $2'$), this is sufficient.
In this section we first define the correct set C (Section 6.1), discuss a central property of Algorithms P and P' (Section 6.2), then show that Algorithms 1 and $1'$ stabilize C (Section 6.3) and finally prove that Algorithms 1 and $1'$ are inverses (Section 6.4).

6.1 The correct set

In this subsection, we define a set C that contains the set of all triples that can be obtained as intermediate triples, applying Algorithms 2 and $2'$. (In fact, C is exactly the set of all intermediate triples.)

Definition 6.1 *Let C be the set of all triples $\big(A, f, (i_0, j_0)\big)$ such that:*

- (i_0, j_0) *are the coordinates of a cell of A,*
- A *is ordered up to (i_0, j_0),*
- f *is a hook function such that $f(i, j) = 0$ for all (i, j) strictly greater than (i_0, j_0),*
- *All candidate elements p' associated with the triple $\big(A, f, (i_0, j_0)\big)$ move to position (i_0, j_0) when applying Algorithm P' to $\big(A, p', (i_0, j_0)\big)$.*

To prove that C contains the set of all triples that can be obtained by successive applications of Algorithm 1 to a triple consisting of a tableau, a hook function equal to 0 everywhere and the least cell of the tableau, one has to establish two assertions: the first one is that the triple consisting of a tableau, a hook function equal to zero everywhere and the smallest cell of the tableau belongs to C. The second one is that Algorithm 1 sends each element of C to an element of C. The first statement is clear and one can see that the triple obtained by applying Algorithm 1 to an element of C satisfies the first *three* conditions of the previous definition as in Note 4.3. We will consider the fourth condition in Section 6.3.
Similarly, to prove that C contains the set of all triples that can be obtained by successive applications of Algorithm $1'$ to a triple consisting of a standard tableau, a hook function, and $(1, 1)$, one has to establish two things: the first is that the triple consisting of a standard tableau, a hook function and $(1, 1)$ belongs to C. The second one is that Algorithm $1'$ maps C to C. The first assertion is obvious and one can see that the triple obtained by applying Algorithm $1'$ to an element of C satisfies the first *two* conditions of the previous definition: the first one is clear and the second one is true as long as condition 4 holds. We will look at the last two conditions in Section 6.3.

6.2 Paths and reverse paths

In this subsection, we define the path of an element, the reverse path of an element and establish a relationship between them. It is this property that makes everything work well in our algorithms.

Definition 6.2 *The* path *of an element a in a tableau T is the set of the cells that a passes through when applying Algorithm P to (T, a). The* reverse path *of an element a' with respect to a cell (i'_0, j'_0) in tableau T' is the set of the cells that a' passes through when applying Algorithm P' to $(T', a', (i'_0, j'_0))$.*

In the following example, we show three successive applications of Algorithm 1: the input triples consist of a tableau, a hook function (given by its first column, the other columns being irrelevant) and the cell (i_0, j_0) whose element is underlined. The elements over the arrows are those in $s(i_0, j_0)$. The boldface elements are the candidates associated with the tableau and its underlined cell. The tableaux are labeled T_1 to T_4 in order of application of the algorithm.

$$
\begin{array}{|c|c|c|c|}\hline 19 & 1 & 4 & 9 \\ \hline 20 & 2 & 8 & 11 \\ \hline 5 & 3 & 10 & 12 \\ \hline \underline{6} & \mathbf{7} & 14 & 15 \\ \hline 13 & 16 & 17 & \mathbf{18} \\ \hline \end{array}
\;,\;
\begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 1 \\ \hline 3 \\ \hline \end{array}
\quad \xrightarrow{5} \quad
\begin{array}{|c|c|c|c|}\hline 19 & 1 & 4 & 9 \\ \hline 20 & 2 & 8 & 11 \\ \hline \underline{3} & \mathbf{5} & 10 & 12 \\ \hline 6 & \mathbf{7} & 14 & 15 \\ \hline 13 & 16 & 17 & \mathbf{18} \\ \hline \end{array}
\;,\;
\begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline 1 \\ \hline 1 \\ \hline 3 \\ \hline \end{array}
$$

$$
\xrightarrow{20} \quad
\begin{array}{|c|c|c|c|}\hline 19 & 1 & 4 & 9 \\ \hline \underline{\mathbf{2}} & 5 & 8 & 11 \\ \hline \mathbf{3} & 7 & 10 & 12 \\ \hline 6 & 14 & \mathbf{15} & 18 \\ \hline 13 & 16 & 17 & \mathbf{20} \\ \hline \end{array}
\;,\;
\begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 2 \\ \hline 3 \\ \hline \end{array}
\quad \xrightarrow{19} \quad
\begin{array}{|c|c|c|c|}\hline \underline{1} & 4 & 8 & 9 \\ \hline 2 & 5 & 10 & 11 \\ \hline 3 & \mathbf{7} & 12 & 18 \\ \hline 6 & 14 & 15 & \mathbf{19} \\ \hline 13 & 16 & 17 & \mathbf{20} \\ \hline \end{array}
\;,\;
\begin{array}{|c|}\hline -1 \\ \hline -1 \\ \hline 1 \\ \hline 3 \\ \hline 3 \\ \hline \end{array}
$$

In our example, the path of 19 in T_3 is the set $\{(1,1), (1,2), (1,3), (2,3), (3,3), (3,4), (4,4)\}$. The reverse path of 20 in T_3 is the set $\{(2,1), (2,2), (3,2), (4,2), (4,3), (4,4), (5,4)\}$. This set is also the set of cells of the path of 20 in T_2.

Definition 6.3 *Let us consider the path π of an element a in cell (i_0, j_0). We say that an element a' is to the* left of the path π *if the bottom-most row that contains an element of π and an element of the reverse path π' of a' with respect to $s^{-1}(i_0, j_0)$ satisfies: every cell of π' is weakly left of every cell of π. If such a row does not exist or if the condition is not satisfied, we say that a' is* to the right *of π. Similarly we say a cell is to the left or right of π if the element in it is.*

For example, in T_3, all the elements are to the left of the path of 19 except 1, 4, 9, 11 and 12. In T_2, 19 is to the right of the path of 20 since such a row does not exist. We are now in position to prove our main result about paths and reverse paths.

Theorem 6.4 *Let $\big(A, f, (i_0, j_0)\big)$ be an element of C and π be the path of $A(s(i_0, j_0))$ in A. If $a' \leq n$ and π' is the reverse path of $(a', (i_0, j_0))$ in A, then all the elements of π' except possibly $s(i_0, j_0)$ (if it is on π') are on the same side of π.*

Proof — First, notice that if $i_0 = 1$, the theorem is obvious since all elements are to the right of π except element weakly left of $s(i_0, j_0)$. So we can assume that $i_0 \neq 1$. Let us now consider a cell (i, j) containing element δ. If it is the only element of its reverse path π', the theorem is obvious. Otherwise, $(i-1,j)$ or $(i,j-1)$ belongs to π'. If we prove that the cell containing the larger element is on the same side of π as δ, then the theorem holds by induction on the length of π'. If $(i-1,j-1)$ does not belong to π then the assertion holds: both cells are on the same side of π as (i,j) except in the case where $(i,j-1)=s(i_0,j_0)$ as noted in the statement of the Theorem. So we assume that $(i-1,j-1)$ belongs to π. If it is the last cell of π, the assertion is also true: by definition both (i,j) and its larger neighbor are on the same side of π. If this is not the case, $(i-1,j)$ or $(i,j-1)$ belong to π. Depending on which content of these cells is larger, we are in one of two cases (the cell (i,j) is the bottom-right one and $\alpha<\beta<\gamma<\delta$):

α	β
γ	δ

α	γ
β	δ

Using the exchanging rules (see Step 2 in Algorithms P and P'), we deduce that in the first case, δ is to the left of π and so is γ. In the second case, δ is to the right of π and so is γ. So, we have proved that in any case, δ and its greatest neighbor are on the same side of π. □

Corollary 6.5 *Using the same notation as in Theorem 6.4. Assume that a' is in a cell different from $s(i_0,j_0)$ and that $i_0\neq 1$. Then a' is to the left of π if and only if (i_0,j_0) belongs to π'.*

Proof— Since (i_0,j_0) is to the left of π for $i_0\neq 1$, it is clear that if (i_0,j_0) belongs to π', a' is to the left of π. Conversely, if a' is to the left of π, notice that $s(i_0,j_0)=(i_0-1,j_0)$ belongs to π so that (i_0-1,j_0) is the only cell of its row that is to the left of π. Since a' is different from (i_0-1,j_0) and to the left of π, (i_0,j_0) or (i_0-1,j_1) for $j_1>j_0$ belong to π'. But the second cell is to the right of π. Theorem 6.4 then implies that (i_0,j_0) belongs to π'. □

For example, 14 and 15 are to the right of the path of 20 in T_2, 15 and 20 are to the left of the path of 19 in T_3.

Corollary 6.6 *Let $\big(A,f,(i_0,j_0)\big)$ be an element of C and let π be the path of $A(s(i_0,j_0))$ in A. Then all the candidates are to the left of π.*

Proof— This is a direct consequence of Corollary 6.5 and property 4 of the definition of C. □

6.3 Both algorithms stabilize C

In this subsection, we prove that both algorithms stabilize C, that is, if a triple belongs to C, its image under each algorithm also belongs to C. We begin with Algorithm 1.

Theorem 6.7 *Algorithm 1 stabilizes C.*

Proof— Let $\big(A,f,(i_0,j_0)\big)$ be an element of C, and let $\big(B,g,s(i_0,j_0)\big)$ be its image under Algorithm 1. In Section 6.1, we saw that $\big(B,g,s(i_0,j_0)\big)$ satisfies the first three conditions of C. We are going to

prove that it satisfies the fourth one. Notice first that if $i_0 = 1$, the theorem is obvious since there is only one candidate: $A(s(i_0, j_0))$. So we can assume that $i_0 \neq 1$. By Step 2 of Algorithm 1, we know that $A(s(i_0, j_0)) = a$ is a candidate that moves to $s(i_0, j_0)$ when applying Algorithm P'. The other candidates are in rows below or above the row of a in B. Since the hook function did not change in the rows below a, these candidates are the same for B as they are for A. Until they reach the row of a, their reverse path is not changed. Since they were to the left of the path of a, they reach this row weakly left of the position of a in B. It is then clear that these elements are to the left of the path of $B(s(i_0, j_0))$ since this is the case for a (see Theorem 6.4).
Consider now a candidate of B that is in a row above the row of a. Denote its cell by (i, j). By Step 2 of Algorithm 1, we know that there is a candidate in the cell $(i+1, j+1)$ of A. Since this candidate is to the left of the path of a in A, we deduce that (i, j) is strictly left of the rightmost element of the reverse path of a in B with row j. As before, we can conclude that this element is to the left of the path of $B(s(i_0, j_0))$. Thanks to Corollary 6.5, the theorem is proved. □

We now come to Algorithm $1'$.

Theorem 6.8 *Algorithm* $1'$ *stabilizes* C.

Proof — Let $\big(A, f, (i_0, j_0)\big)$ be an element of C, and let $\big(A', f', s^{-1}(i_0, j_0)\big)$ be its image under Algorithm $1'$. Thanks to Section 6.1, we know that $\big(A', f', s^{-1}(i_0, j_0)\big)$ satisfies the first two conditions of C. Let us look at the third one. Consider the maximum candidate a associated with A. Any candidate b that is strictly above it, say in cell (i, j), is also strictly to its left: consider the southeast-most cell that belongs to both reverse paths. Note that the paths must agree from this cell to (i_0, j_0). Since the coding of a is greater than the coding of b, and since the path of a cannot stop on this cell, its next move is necessarily east (W in the coding). Because a's path reaches a lower row than b's and they never intersect again (we took the southeast-most cell), b is necessarily strictly to the left of (i, j). This shows that the cell $(i+1, j+1)$ belongs to the diagram. So, the new value of the hook function at a cell never exceeds the number of cells to the right of the given cell.
We now consider the fourth condition. The proof is very similar to the proof of the previous theorem. First, if (i_0, j_0) is at the bottom of its column, the fourth condition is obviously satisfied since the new candidates necessarily move to $s^{-1}(i_0, j_0)$. So, we can assume that (i_0, j_0) is not at the bottom of its column. Denote by π the path of $A'(i_0, j_0)$ in A' and note that it is the same set of cells as the reverse path of $A'(i_0, j_0)$ in A. The hook function values do not change for the candidates that are below the last row of π so they remain to the left of π. For the candidates above the last row of π, the idea is the same as before: since $A'(i_0, j_0)$ has the greatest coding, the candidates of A that are on π are followed by an east move (W in the coding) so the corresponding candidates in A' are to the left of π. The idea is the same for the elements that are strictly at the left of the reverse path of $A'(i_0, j_0)$ in A. By Corollary 6.5, we are done. □

6.4 The algorithms are inverse to each other

In this subsection, we prove that Algorithm 1 is the inverse of Algorithm $1'$.
We first prove that Algorithm 1 is the left inverse of Algorithm $1'$.

Theorem 6.9 *Let $\big(A, f, (i_0, j_0)\big)$ be a triple belonging to C. Applying Algorithm $1'$ to it, we obtain $\big(A', f', s^{-1}(i_0, j_0)\big)$. Applying Algorithm 1 to this triple, we obtain $\big(B', g', (i_0, j_0)\big)$. Then $B' = A$ and $g' = f$.*

Proof — First, it is clear that $B' = A$: we apply Algorithm 1 to the integer that was the maximum candidate of A and the cells of its path are exactly the cells of its reverse path in A. Since $B' = A$, we obtain directly that $g' = f$ since the respective changes on the hook functions are the inverse of each other and depend only on the initial and final cells of the path. □

We now prove that Algorithm 1 is the right inverse of Algorithm $1'$.

Theorem 6.10 *Let $\big(A, f, (i_0, j_0)\big)$ be a triple belonging to C. Applying Algorithm 1 to it, we obtain $\big(B, g, s(i_0, j_0)\big)$. Applying Algorithm $1'$ to this triple, we obtain $\big(B', g', (i_0, j_0)\big)$. Then $B' = A$ and $g' = f$.*

Proof — First, we prove that $B' = A$. Notice that we already proved (see the proof of Theorem 6.7) that the codings of the other candidates are smaller than the coding of $A(s(i_0, j_0))$, since their coding is either followed by a W move in π or they differ in a cell where their coding is N. So the maximum candidate is $A(s(i_0, j_0))$. Since the cells of its reverse path in B are the same as the cells of its path in A, we obtain $B' = A$. The same argument as before allows us to conclude that $g' = f$. □

7 Examples

7.1 Example 1

We give an example of how Algorithm 2 works. The element under consideration when applying Algorithm 1 is underlined.

$$
\begin{array}{|c|c|c|}\hline 10 & 11 & 5 \\ \hline 7 & 9 & 6 \\ \hline 2 & 1 & \underline{4} \\ \hline 8 & 3 \\ \cline{1-2}\end{array}\ ,\
\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2}\end{array}
\xrightarrow{6}
\begin{array}{|c|c|c|}\hline 10 & 11 & 5 \\ \hline 7 & 9 & \underline{4} \\ \hline 2 & 1 & 6 \\ \hline 8 & 3 \\ \cline{1-2}\end{array}\ ,\
\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 0 & -1 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2}\end{array}
$$

$$
\xrightarrow{5}
\begin{array}{|c|c|c|}\hline 10 & 11 & \underline{4} \\ \hline 7 & 9 & 5 \\ \hline 2 & 1 & 6 \\ \hline 8 & 3 \\ \cline{1-2}\end{array}\ ,\
\begin{array}{|c|c|c|}\hline 0 & 0 & -2 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2}\end{array}
\xrightarrow{3}
\begin{array}{|c|c|c|}\hline 10 & 11 & 4 \\ \hline 7 & 9 & 5 \\ \hline 2 & 1 & 6 \\ \hline 8 & \underline{3} \\ \cline{1-2}\end{array}\ ,\
\begin{array}{|c|c|c|}\hline 0 & 0 & -2 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2}\end{array}
$$

$$
\xrightarrow{1}
\begin{array}{|c|c|c|}\hline 10 & 11 & 4 \\ \hline 7 & 9 & 5 \\ \hline 2 & \underline{1} & 6 \\ \hline 8 & 3 \\ \cline{1-2}\end{array}\ ,\
\begin{array}{|c|c|c|}\hline 0 & 0 & -2 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2}\end{array}
\xrightarrow{9}
\begin{array}{|c|c|c|}\hline 10 & 11 & 4 \\ \hline 7 & \underline{1} & 5 \\ \hline 2 & 3 & 6 \\ \hline 8 & 9 \\ \cline{1-2}\end{array}\ ,\
\begin{array}{|c|c|c|}\hline 0 & 0 & -2 \\ \hline 0 & -1 & 0 \\ \hline 0 & -1 & 0 \\ \hline 0 & 0 \\ \cline{1-2}\end{array}
$$

$$\xrightarrow{11} \begin{array}{|c|c|c|}\hline 10 & \underline{1} & 4 \\ \hline 7 & 3 & 5 \\ \hline 2 & 6 & 11 \\ \hline 8 & 9 \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & -2 & -2 \\ \hline 0 & -2 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array} \quad \xrightarrow{8} \begin{array}{|c|c|c|}\hline 10 & 1 & 4 \\ \hline 7 & 3 & 5 \\ \hline 2 & 6 & 11 \\ \hline \underline{8} & 9 \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & -2 & -2 \\ \hline 0 & -2 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array}$$

$$\xrightarrow{2} \begin{array}{|c|c|c|}\hline 10 & 1 & 4 \\ \hline 7 & 3 & 5 \\ \hline \underline{2} & 6 & 11 \\ \hline 8 & 9 \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & -2 & -2 \\ \hline 0 & -2 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array} \quad \xrightarrow{7} \begin{array}{|c|c|c|}\hline 10 & 1 & 4 \\ \hline \underline{2} & 3 & 5 \\ \hline 6 & 7 & 11 \\ \hline 8 & 9 \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & -2 & -2 \\ \hline -1 & -2 & 0 \\ \hline 1 & 1 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array}$$

$$\xrightarrow{10} \begin{array}{|c|c|c|}\hline \underline{1} & 3 & 4 \\ \hline 2 & 5 & 10 \\ \hline 6 & 7 & 11 \\ \hline 8 & 9 \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline -2 & -2 & -2 \\ \hline 2 & -2 & 0 \\ \hline 1 & 1 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array}$$

7.2 Example 2

We now give an example of how Algorithm $2'$ works. The candidate elements are boldfaced and the content of the considered cell when applying Algorithm $1'$ is underlined.

$$\begin{array}{|c|c|c|}\hline \underline{1} & \mathbf{2} & 7 \\ \hline 3 & 4 & \mathbf{8} \\ \hline 5 & 9 & 10 \\ \hline 6 & \mathbf{11} \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 1 & 1 & -2 \\ \hline 2 & -1 & 0 \\ \hline -1 & 1 & 0 \\ \hline 1 & 0 \\ \cline{1-2} \end{array} \quad \xrightarrow{8} \begin{array}{|c|c|c|}\hline 8 & 1 & 2 \\ \hline \underline{3} & 4 & \mathbf{7} \\ \hline 5 & 9 & 10 \\ \hline 6 & \mathbf{11} \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & 1 & -2 \\ \hline 2 & -1 & 0 \\ \hline -1 & 1 & 0 \\ \hline 1 & 0 \\ \cline{1-2} \end{array}$$

$$\xrightarrow{7} \begin{array}{|c|c|c|}\hline 8 & 1 & 2 \\ \hline 7 & 3 & 4 \\ \hline \underline{5} & 9 & 10 \\ \hline 6 & \mathbf{11} \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & 1 & -2 \\ \hline 0 & -1 & 0 \\ \hline -1 & 1 & 0 \\ \hline 1 & 0 \\ \cline{1-2} \end{array} \quad \xrightarrow{11} \begin{array}{|c|c|c|}\hline 8 & 1 & 2 \\ \hline 7 & 3 & 4 \\ \hline 11 & 5 & 10 \\ \hline \underline{\mathbf{6}} & 9 \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & 1 & -2 \\ \hline 0 & -1 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array}$$

$$\xrightarrow{6} \begin{array}{|c|c|c|}\hline 8 & \underline{1} & \mathbf{2} \\ \hline 7 & 3 & 4 \\ \hline 11 & 5 & \mathbf{10} \\ \hline 6 & \mathbf{9} \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & 1 & -2 \\ \hline 0 & -1 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array} \quad \xrightarrow{2} \begin{array}{|c|c|c|}\hline 8 & 2 & 1 \\ \hline 7 & \underline{3} & 4 \\ \hline 11 & 5 & \mathbf{10} \\ \hline 6 & \mathbf{9} \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & 0 & -2 \\ \hline 0 & -1 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array}$$

$$\xrightarrow{10} \begin{array}{|c|c|c|}\hline 8 & 2 & 1 \\ \hline 7 & 10 & 4 \\ \hline 11 & \underline{\mathbf{3}} & 5 \\ \hline 6 & \mathbf{9} \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & 0 & -2 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array} \quad \xrightarrow{3} \begin{array}{|c|c|c|}\hline 8 & 2 & 1 \\ \hline 7 & 10 & 4 \\ \hline 11 & 3 & 5 \\ \hline 6 & \underline{\mathbf{9}} \\ \cline{1-2} \end{array}\ ,\ \begin{array}{|c|c|c|}\hline 0 & 0 & -2 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array}$$

$$\xrightarrow{9} \begin{array}{|c|c|c|}\hline 8 & 2 & \underline{1} \\ \hline 7 & 10 & \mathbf{4} \\ \hline 11 & 3 & \mathbf{5} \\ \hline 6 & 9 \\ \cline{1-2} \end{array} \; , \; \begin{array}{|c|c|c|}\hline 0 & 0 & -2 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array} \quad \xrightarrow{4} \begin{array}{|c|c|c|}\hline 8 & 2 & 4 \\ \hline 7 & 10 & \underline{1} \\ \hline 11 & 3 & \mathbf{5} \\ \hline 6 & 9 \\ \cline{1-2} \end{array} \; , \; \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 0 & -1 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array}$$

$$\xrightarrow{5} \begin{array}{|c|c|c|}\hline 8 & 2 & 4 \\ \hline 7 & 10 & 5 \\ \hline 11 & 3 & \underline{\mathbf{1}} \\ \hline 6 & 9 \\ \cline{1-2} \end{array} \; , \; \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 & 0 \\ \hline 0 & 0 \\ \cline{1-2} \end{array}$$

References

[1] J. S. Frame, G. de B. Robinson, R. M. Thrall, "The hook graphs of the symmetric group," *Canad. J. Math.* **6** (1954), 316–325.

[2] D. S. Franzblau, D. Zeilberger, "A bijective proof of the hook-length formula," *J. Algorithms* **3** (1982), 317–343.

[3] C. Greene, A. Nijenhuis, H. S. Wilf, "A probabilistic proof of a formula for the number of Young tableaux of a given shape," *Adv. in Math.* **31** (1979), 104–109.

[4] A. P. Hillman, R. M. Grassl, "Reverse plane partitions and tableau hook numbers," *J. Combin. Theory Ser A* **21** (1976), 216–221.

[5] G. D. James, A. Kerber, *The Representation Theory of the Symmetric Group*, Encyclopedia of Mathematics and its Applications, Vol. 16, Addison Wesley, Reading, MA, 1981.

[6] C. Krattenthaler, "Bijective proofs of the hook formulas for the number of the standard Young tableaux, ordinary and shifted," *Electronic J. Combinatorics* **2**, (1995), #R13.

[7] I. G. Macdonald, *Symmetric Functions and Hall Polynomials*, 2nd edition, Clarendon Press, Oxford, 1995.

[8] I. M. Pak, A. V. Stoyanovskii, "A bijective proof of the hook-length formula," *Funct. Anal. Appl.* **24**, (1992).

[9] B. E. Sagan, "The ubiquitous Young tableau," in *Invariant Theory and Tableaux*, D. Stanton ed., Springer-Verlag, New York, (1990), 262–298.

[10] B. E. Sagan, *The Symmetric Group*, Wadsworth and Brooks/Cole, California, 1991.

[11] M. P. Schützenberger, "La correspondance de Robinson," in *Combinatoire et représentation du groupe symétrique*, D. Foata ed., Lecture Notes in Math., Vol. 579, Springer-Verlag, New York, NY, (1977), 59–135.

[12] R. Stanley, "Ordered structures and partitions," *Memoirs Amer. Soc.* **119** (1972).

[13] D. Zeilberger, "A short hook-length bijection inspired by the Greene-Nijenhuis-Wilf proof," *Discrete Math.* **51** (1984), 101–108.