

# ALGORITHMS FOR DISCOVERING AND PROVING THEOREMS ABOUT PERMUTATION PATTERNS

HJALTI MAGNUSSON AND HENNING ULFARSSON

**ABSTRACT.** We present an algorithm, called BiSC, that describes the patterns avoided by a given set of permutations. It automatically conjectures the statements of known theorems such as the descriptions of stack-sortable (Knuth 1975) and West-2-stack-sortable permutations (West 1990), smooth (Lakshmibai and Sandhya 1990) and forest-like permutations (Bousquet-Mélou and Butler 2007), and simsun permutations (Brändén and Claesson 2011). The algorithm has also been used to discover new theorems and conjectures related to Young tableaux, Wilf-equivalences and sorting devices. We further give algorithms to prove a complete description of preimages of pattern classes under certain sorting devices. These generalize an algorithm of Claesson and Ulfarsson (2012) and allow us to prove a linear time algorithm for finding occurrences of the pattern 4312.

## CONTENTS

1. Introduction	1
2. Learning mesh patterns	2
3. Sorting algorithms and preimages	8
Acknowledgments	12
References	12

## 1. INTRODUCTION

A *permutation* of length  $n$  is a bijection from the set  $\{1, \dots, n\}$  to itself. We write permutations in the *one-line notation*, where  $\pi_1\pi_2\cdots\pi_n$  is the permutation that sends  $i$  to  $\pi_i$ . If  $w = w_1w_2\cdots w_k$  is a word of distinct integers,  $\text{fl}(w)$  is the permutation obtained by replacing the  $i$ th smallest letter in  $w$  with  $i$ . This is called the *flattening* of  $w$ . A permutation  $\pi$  of length  $n$  *contains* a permutation  $p$  of length  $k$  if there exist indices  $1 \leq j_1 < j_2 < \cdots < j_k \leq n$  such that  $\text{fl}(\pi_{j_1}\pi_{j_2}\cdots\pi_{j_k}) = p$ . In this context  $p$  is called a (*classical*) *pattern*. If  $\pi$  does not contain  $p$  then it *avoids*  $p$ . We let  $\text{Av}(P)$  denote the set of permutations that avoid all the patterns in a set  $P$ .

This extended abstract summarizes two papers, [Ulf12a] and [MU12], which treat algorithms and permutation patterns. The first paper introduces an algorithm, BiSC, which

---

*Key words and phrases.* Permutation Patterns, Sorting algorithms.

The authors are supported by grant no. 090038013-4 from the Icelandic Research Fund.

was inspired by a question posed by Billey [Bil11]. Its input is a set of permutations and its output is a set of mesh patterns that the permutations avoid. Mesh patterns are a type of generalized pattern introduced by Brändén and Claesson [BC11]. BiSC can rediscover the statements of well-known theorems describing properties of permutations with patterns. A few examples may be seen in Tab. 1.

Property	Forbidden patterns	Reference
stack-sortable	231	[Knu75, Section 2.2.1, Exc. 4]
West-2-stack-sortable	2341, (3241, $\{(1, 4)\}$ )	[Wes90, Thm. 4.2.18]
simsun	(321, $\{(1, 0), (1, 1), (2, 2)\}$ )	[BC11, p. 7]
smooth	1324, 2143	[LS90, Thm. 1]
forest-like	1324, (2143, $\{(2, 2)\}$ )	[BMB07, Thm. 1]

TABLE 1. Some statements of known theorems BiSC can rediscover

We emphasize that BiSC discovers statements like the ones in Tab. 1, but does not supply a proof. In subsection 2.2 we present new theorems found by us and others using BiSC.

Section 3 summarizes the second paper [MU12], on algorithms that prove a complete description of preimages of classical pattern classes under a sorting operator. These are based on algorithms in [CU12] for the stack-sorting and bubble-sorting operator. We give a common generalization to a stack of any depth, as well as algorithms for sorting with a queue, a pop stack, with insertion and with the pancake sorting method. We only describe the first two in this extended abstract. We give a linear time (in the size of the input permutation) algorithm for recognizing the pattern 4312. This algorithm was discovered with BiSC and proven with the preimage algorithms for a queue and a stack. It is the first linear algorithm for recognizing a pattern of length greater than 3 (besides the increasing and decreasing patterns). Finally, we extend the preimage algorithm for a stack to preimages of certain mesh pattern classes, enabling us to give a fully automatic proof of the description of West-3-stack-sortable permutations, first done in [Ulf12b].

The algorithms treated here have been implemented in the computer algebra system Sage and are available at <http://staff.ru.is/henningu/programs/pattalgos2012/pattalgos2012.html>.

## 2. LEARNING MESH PATTERNS

Given a set of permutations  $A$ , we call a set of patterns  $b$  a *base* for  $A$  if  $A = \text{Av}(b)$ . There is a well-known algorithm that can find bases consisting only of classical patterns, which proceeds as follows if the input is the set of permutations below.

1, 12, 21, 123, 132, 213, 312, 321, 1234, 1243, 1324, 1423, 1432, 2134, 2143, 3124, 3214, 4123, 4132, 4213, 4321.

The first missing permutation is 231 so we add it to our potential base  $b = \{231\}$ . When we reach the permutations of length 4 we see that 1324, 2314, 2341, 2413, 2431, 3142, 3241, 3412, 3421, 4231 and 4312 are all missing. All of these, except the last, contain the pattern in the base so we should expect them to be missing. The last one, the permutation

4312, does not contain the pattern 231 so we extend the base,  $b = \{231, 4312\}$ . If the input had permutations of length 5 we would continue in the same manner: checking whether the missing permutations contain a previously forbidden pattern, and if not, extend the base by adding new classical patterns.

As the next example shows, we must also make sure that no permutation in the input contains a previously forbidden pattern. Consider the West-2-stack-sortable permutations [Wes90]

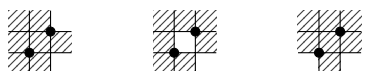
1, 12, 21, 123, 132, 213, 231, 312, 321, 1234, 1243, 1324, 1342, 1423, 1432, 2134, 2143, 2314, 2413, 2431, 3124, 3142, 3214, 3412, 3421, 4123, 4132, 4213, 4231, 4312, 4321,  $\dots$ , 35241,  $\dots$

The first missing permutations are 2341 and 3241 so  $b = \{2341, 3241\}$ . There are many missing permutations of length 5 and all of them are consequences of 2341 being forbidden, e.g., 34152 is not in the input since the subword 3452 is an occurrence of 2341. But one of the input permutations is 35241, which contains the previously forbidden pattern 3241. It seems that the presence of the 5 inside the occurrence of 3241 in 35241 is important. Here we need the notion of mesh patterns [BC11], which allow us to forbid letters from occupying certain regions in a pattern. We must find a shading  $R$  such that the mesh pattern  $(3241, R)$  is contained in the permutation 3241 but avoided by 35241. In this case it is easy to guess  $R = \{(1, 4)\}$  (the top-most square between 3 and 2), which in fact is the correct choice: the base  $b = \{2341, (3241, \{(1, 4)\})\}$  exactly describes the West-2-stack-sortable permutations.

Now consider a more difficult input,

$$1, 21, 321, 2341, 4123, 4321. \quad (1)$$

When we see that the permutation 12 is missing we add the mesh pattern  $(12, \emptyset)$  to our base. The only permutation of length 3 in the input is 321, and this is consistent with our current base. We see that every permutation of length four is forbidden up to the permutation 2341, which makes sense since each of those contains  $(12, \emptyset)$ . But 2341 also contains 12, in fact it contains several occurrences of it: 23, 24 and 34. These occurrences tell us that the following mesh patterns are actually allowed.



$$(2)$$

It is tempting to guess that what is actually forbidden is  $\begin{array}{|c|c|} \hline \bullet & \bullet \\ \hline \bullet & \bullet \\ \hline \end{array}$ . We are forced to revisit all the permutations that are not in the input to check whether they contain our modified forbidden pattern. This fails for the permutation 231. We must add something to our base, but what? Based on the mesh patterns in (2) we could add  $\begin{array}{|c|c|} \hline \bullet & \bullet \\ \hline \bullet & \bullet \\ \hline \end{array}$ , which is contained in 231 but not contained in any permutation in the input we have looked at up to now. From this small example it should be clear that it quickly becomes difficult to go back and forth in the input, modifying the currently forbidden patterns, making sure that they are not contained in any permutation in the input, while still being contained in the permutations not in the input. We need a more unified approach. But first, if the reader

is curious the permutations in (1) are the permutations of length at most 4 in the set

$$\text{Av} \left( \begin{array}{|c|c|} \hline \text{+} & \text{+} \\ \hline \text{+} & \text{+} \\ \hline \end{array}, \begin{array}{|c|c|} \hline \text{+} & \text{+} \\ \hline \text{+} & \text{+} \\ \hline \end{array} \right). \quad (3)$$

**2.1. The BiSC algorithm.** In the last example a mesh pattern with the underlying classical pattern 12 first appeared in a permutation of length 4. This motivates the first step in the high-level description of BiSC:

- (1) Search the permutations in the input and record which mesh patterns are allowed
- (2) Infer the forbidden patterns from the allowed patterns found in step (1)

More precisely, step (1) is accomplished with Algorithm 1.

**Input:**  $A$ , a finite set of permutations; and  $m$ , an upper bound on length of the patterns to search for

**Output:** A list,  $S$ , consisting of  $(p, \text{sh}_p)$  for all classical patterns of length at most  $m$

```

1 Initialize  $S = \{(p, \emptyset) : p \text{ classical pattern of length at most } m\}$ 
2 for  $\pi \in A$  do
3   for  $s \in \text{subwords}_{\leq m}(\pi)$  do
4     Let  $p = \text{fl}(s)$ 
5     Let  $R$  be the maximal shading of  $p$  for the occurrence  $s$  in  $\pi$ 
6     if  $R \not\subseteq T$  for all shadings  $T \in \text{sh}_p$  then
7       Add  $R$  to  $\text{sh}_p$ 
```

**Algorithm 1:** Mine

In line 3  $\text{subwords}_{\leq m}(\pi)$  is the set of (not necessarily consecutive) subwords of length at most  $m$  in the permutation  $\pi$ , e.g.,  $\text{subwords}_{\leq 2}(1324) = \{13, 12, 14, 32, 34, 24, 1, 3, 2, 4\}$ . In line 5 we let  $R$  be the maximal shading that can be applied to the classical pattern  $p$  while ensuring that  $s$  is still an occurrence of  $(p, R)$ , see Fig. 1.

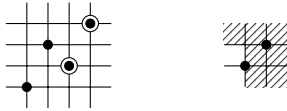




FIGURE 1. Applying the maximal shading to the classical pattern 12 so that  $s = 24$  is still an occurrence in 1324

In line 6,  $\text{sh}_p$  is a set we maintain of maximal shadings of  $p$ . For example if we see that  is an allowed shading, and then later come across  we will only store the second shading, since the first one has now become redundant.

**Lemma 2.1.** *If  $(p, R)$  is any mesh pattern, of length not longer than  $m$ , that occurs in a permutation in  $A$  then there exists a  $(p, R')$  in the output of  $\text{Mine}(A, m)$  such that  $R \subseteq R'$ .*

**Lemma 2.2.** *If a set of permutations  $A$  is defined by the avoidance of a (possibly infinite) list of classical patterns  $P$  and  $(q, \text{sh}_q)$  with  $\text{sh}_q \neq \emptyset$  is in the output of  $\text{Mine}(A_{\leq n}, m)$  then  $q$  is not in the list  $P$  and  $\text{sh}_q$  only contains the full shading of  $q$ .*

*Proof.* If  $\text{sh}_q$  does not contain the complete shading then  $q$  is not in  $A$ , but appears as a classical pattern in some larger permutation  $\pi$  in  $A$ . Since  $q$  is not in  $A$  it must contain one of the forbidden patterns  $p$  in  $P$ , which would imply that  $p$  also occurs in  $\pi$ , contradicting the fact that  $\pi$  is in  $A$ .  $\square$

Step (2) in the high-level description is implemented in Algorithm 2, where we generate the forbidden patterns from the allowed patterns in the output of **Mine**.

**Input:**  $S$ , a list of classical patterns along with shadings  $\text{sh}_p$

**Output:** A list consisting of  $(p, \text{forb}_p)$  where  $p$  is a classical pattern and  $\text{forb}_p$  is a set of minimal forbidden shadings

```

1 for  $(p, \text{sh}_p) \in S$  do
2   Let  $\text{forb}_p$  be the minimal shadings of  $p$  that are not contained in any member of
    $\text{sh}_p$ 
3   for  $R \in \text{forb}_p$  do
4     if  $R$  is a consequence of some shading in  $\text{forb}_q$  for a pattern  $q$  contained in  $p$ 
     then
5       Remove  $R$  from  $\text{forb}_p$ 

```

### Algorithm 2: Forb

To explain lines 4–5 in the algorithm assume  $p = 1243$  and the shading,  $R$ , on the left in Fig. 2 is one of the shadings in  $\text{forb}_p$  on line 2. Assume also that earlier in the outer-most for-loop we generated  $q = 12$  with the shading,  $R'$  on the right in Fig. 2. Then the shading  $R$  is removed from  $\text{forb}_p$  on line 5 since any permutation containing  $(p, R)$ , also contains  $(q, R')$ .

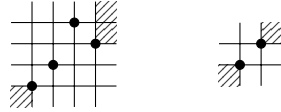


FIGURE 2. The mesh patterns  $(p, R)$  and  $(q, R')$

We now define  $\text{BiSC}(A, m) = \text{Forb}(\text{Mine}(A, m))$ .

**Theorem 2.3.** *Let  $A$  be any set of permutations. Then for all positive integers  $N \geq n$  and  $m$*

$$A_{\leq n} \subseteq \text{Av}(\text{BiSC}(A_{\leq N}, m))_{\leq n}.$$

*Furthermore, if the set  $A$  is defined in terms of a finite list of patterns (classical or not), whose longest pattern has length  $k$ , and if  $N \geq n \geq k$ ,  $m \geq k$  then there is equality between the two sets above.*


*Proof.* To prove the subset relation let  $\pi$  be a permutation in  $A_{\leq n}$  and  $(p, R)$  be a mesh pattern of length at most  $m$  contained in  $\pi$ . By Lemma 2.1 there is a pattern  $(p, R')$  in the output from  $\text{Mine}(A_{\leq N}, m)$  such that  $R \subseteq R'$ . This implies that  $R$  is not one of the shadings in  $\text{forb}_p$  and therefore  $(p, R)$  is not in the output of  $\text{BiSC}(A_{\leq n}, m)$ . To prove

equality of the sets let  $\pi$  be a permutation that is not in the set on the right because it contains a mesh pattern  $(p, R)$  from the list defining  $A$ . If  $p$  never occurs as a classical pattern in a permutation in  $A_{\leq N}$  then  $(p, \emptyset)$  is in the output of  $\text{Mine}(A_{\leq N}, m)$  which implies that  $(p, \emptyset)$  is in the output of  $\text{BiSC}(A_{\leq N}, m)$ . This implies that  $\pi$  is not in the set on the right. If, however,  $p$  occurs in some permutations in  $A_{\leq N}$  then every occurrence of a mesh pattern  $(p, R')$  must satisfy  $R \not\subseteq R'$ . This implies that when  $\text{forb}_p$  is created in line 2 in Algorithm 2 it contains  $R$  or a non-empty subset,  $R''$ , of it. We assume without loss of generality that  $R''$  is not removed due to redundancy in line 5. This implies that  $(p, R'')$  is in the output of  $\text{BiSC}(A_{\leq N}, m)$  so  $\pi$  is not in the set on the right.  $\square$

If the input  $A$  to  $\text{BiSC}$  is defined in terms of classical patterns we can strengthen the previous theorem.

**Theorem 2.4.** *If a set of permutations  $A$  is defined by the avoidance of a (possibly infinite) list  $P$  of classical patterns then the output of  $\text{BiSC}(A_{\leq n}, m)$  for any  $n, m$  will consist only of classical patterns. Furthermore, if the longest patterns in the list  $P$  have length  $k$ , then  $A = \text{Av}(\text{BiSC}(A_{\leq n}, m))$  for any  $n, m \geq k$ .*

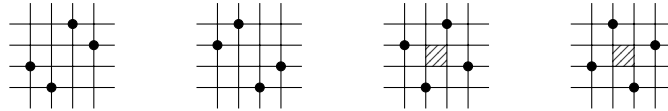
*Proof.* This follows from Lemma 2.2, see [MU12] for the details.  $\square$

Some properties require us to look at very large permutations to discover patterns. For example one must look at permutations of length 8 in the set in (3) above to see that the mesh pattern  is allowed. But as examples in the next section show it often suffices to look at permutations of length  $m + 1$  when searching for mesh patterns of length  $m$ .

For some input sets  $A$  there can be redundancy in the output  $\text{BiSC}(A, m)$ , in the sense that some patterns can be omitted without changing  $\text{Av}(\text{BiSC}(A, m))$ . We propose three ways to fix this in [Ulf12a]. This is never a problem in the following applications so we omit further details.

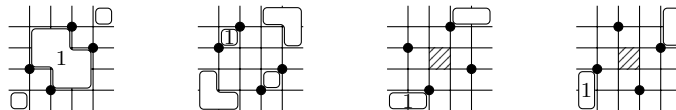
## 2.2. Some applications of $\text{BiSC}$ .

Forbidden shapes in Young tableaux. There is a well-known bijection, called the Robinson-Schensted-Knuth-correspondence (RSK) between permutations of length  $n$  and pairs of Young tableaux of the same shape. Let  $A$  be the set of permutations whose image tableaux are hook-shaped. The output of  $\text{BiSC}(A_{\leq 5}, 4)$  is the following four patterns.



Here we have rediscovered an observation made by Atkinson [Atk98, Proof of Lemma 9] that the avoiders of the four patterns above are exactly the permutations whose tableaux are hook-shaped. We can view the property of being a hook-shaped tableau as not containing the tableaux shape  $(2, 2)$  (two rows with two boxes each). It is natural to ask if we can in general describe the permutations corresponding to tableaux not containing a particular shape  $\lambda$ . E.g., let  $A$  be the permutations whose tableaux avoid  $\lambda = (3, 2)$ . The output of  $\text{BiSC}(A_{\leq 6}, 5)$  is 25 mesh patterns that can be simplified to give the following.

**Proposition 2.5.** *The permutations whose tableaux under the RSK-correspondence avoid the shape  $(3, 2)$  are precisely the avoiders of the marked mesh patterns ([Ulf11, Definition 4.5]) below.*



The meaning of a marked region is that it should contain at least one point, and therefore the first pattern above corresponds to 9 classical patterns.

Crites et al. [CPW11] showed that if a permutation  $\pi$  contains a separable classical pattern  $p$  then the tableaux shape of  $p$  is contained in the tableaux shape of  $\pi$ . Atkinson's observation and the proposition above suggest a more general result holds if one considers mesh patterns instead of classical patterns.

Forbidden tree patterns. Pudwell et al. [DPTW12] used the bijection between binary trees and 231-avoiding permutations and the BiSC algorithm to discover a correspondence between tree patterns and mesh patterns. This can be used to generate Catalan many Wilf-equivalent sets of the form  $\text{Av}(231, p)$  where  $p$  is a mesh pattern. The tree in Fig. 3 gives the Wilf-equivalence between the sets  $\text{Av}(231, 654321)$  and  $\text{Av}(231, (126345, \{(1, 6), (4, 5), (4, 6)\}))$ .

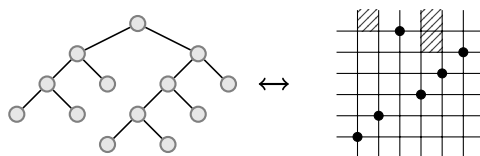
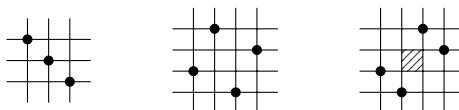


FIGURE 3. A binary tree and its corresponding mesh pattern, see [DPTW12]

1-quicksortable permutations. Consider the following sorting method II: If the input permutation is empty, return the empty permutation. If the input permutation  $\pi$  is not empty and contains strong fixed points, let  $x$  be the right-most such point, write  $\pi = \alpha x \beta$  and recursively apply the method to  $\alpha$  and  $\beta$ . If  $\pi$  does not contain a strong fixed point we move every letter in  $\pi$  that is smaller than the first letter to the start of the permutation. This operator was proposed by Claesson [Cla12] as a single pass of quicksort [Hoa62]. Let  $A$  be the set of permutations that are sortable in at most one pass under this operator. The output of  $\text{BiSC}(A_{\leq 5}, 4)$  is the following.



Stuart Hannah [Han12] verified that the sortable permutations are precisely the avoiders of these three patterns and, furthermore, that they are in bijection with words on the alphabet  $(a + (bb\star)(cc\star))\star$  which are counted by the sequence [Slo12, A034943].

The next section presents algorithms that can prove a complete description of the preimage of pattern classes under two sorting devices: a stack of depth  $d$  and a queue. More devices are treated in [MU12].

### 3. SORTING ALGORITHMS AND PREIMAGES

**3.1. Sorting with a stack of any depth.** A *stack* is a list with the restriction that elements can only be added and removed from one end of the list. We call this end the *top* of the stack. The act of adding an element to a stack is called *pushing*, and the act of removing is called *popping*. The *depth* of a stack is the number of elements it can hold along with one extra space used for passing elements through the stack.

Given an input permutation  $\pi$ , let  $S(\pi)$  be the permutation obtained by the following procedure [Knu75]: (1) If the stack is empty or the topmost element is larger than the first element of the input,  $s$ , push  $s$  onto the stack. (2) Otherwise, pop elements from the stack into the output permutation, until  $s$  can be pushed onto the stack. (3) Repeat this process until the input is empty. (4) Empty the stack into the output permutation.

The same procedure can also be used with stacks of limited depth. We let  $S_d(\pi)$  denote the permutation obtained by applying the procedure, with a stack of depth  $d$ , to a permutation  $\pi$ . It can be easily proven that for a permutation  $\pi = \alpha n \beta$ , where  $n$  is the largest element of  $\pi$ , that

$$S_d(\pi) = S_d(\alpha n \beta) = \begin{cases} S_d(\alpha) S_{d-1}(\beta) n & \text{if } d > 1, \\ \pi & \text{if } d = 1. \end{cases}$$

We call  $S_d$  the *stack-sort operator with a stack of depth  $d$* . If the result of applying  $S_d$  to a permutation  $\pi$  is the identity permutation, we say that  $\pi$  is *stack-sortable with a stack of depth  $d$* .

Note that  $S_\infty = S$  is the stack-sort operator (of unlimited depth), and  $S_2 = B$  is the bubble sort operator (see e.g. [AAB<sup>+</sup>11]). If we apply  $S_3$  to the permutation 45321, then 4 is pushed onto the stack, but immediately popped by the 5. Now we have 4 in the output and 5 on the stack. The 3 is then pushed onto the stack and the stack is now full. Finally 2 and 1 bypass the stack, and the stack is subsequently emptied into the output. Thus  $S_3(45321) = 42135$ .

Algorithm 1 in [CU12] gives a description of the preimage of any set defined by the avoidance of classical patterns for  $S$ . More precisely, given a classical pattern  $p$ , the algorithm outputs a set  $M$  of marked mesh patterns such that

$$\text{Av}(M) = \{\pi : S(\pi) \in \text{Av}(p)\}.$$

The algorithm proceeds in two steps. In the first step classical patterns, called *candidates*, are generated. In the second step, shadings and markings are added to the candidates, to produce the patterns that describe the preimage. Note that it might be impossible to add shadings or markings to some of the candidates produced by the first step. We will now give a generalization of this algorithm to handle the preimage of a stack of any depth.



The analog of [CU12, Proposition 4.1], which generates classical pattern candidates, is the following.

**Proposition 3.1.** *Let  $p = \alpha n \beta$  be a permutation of a finite set of integers where  $n$  is the largest element of  $p$  and  $\alpha = a_1 a_2 \cdots a_i$ . Then*

$$\text{cand}_d(p) = \begin{cases} \bigcup_{j=0}^i \{ \gamma n \delta : \gamma \in \text{cand}_d(a_1 a_2 \cdots a_j), \delta \in \text{cand}_{d-1}(a_{j+1} \cdots a_i \beta) \} & \text{if } d > 1, \\ p & \text{if } d = 1. \end{cases}$$

*contains all classical patterns that can become  $p$  after one pass of  $S_d$ .*

To complete the algorithm, we need the notion of a *decorated pattern* [Ulf12b, Definition 2.2]. As was hinted at in the paper, we need a slight modification of these patterns, where some regions are *required* to contain a pattern. We therefore view a decorated pattern as a 5-tuple  $(p, S, M, D, C)$ , where  $p$  is the underlying classical pattern,  $S$  is the set of shaded squares,  $M$  is the set of markings,  $D$  is the set of avoidance decorations, and  $C$  is the set of containment decorations.

It can be easily seen that if  $\sigma = S_d(\pi)$ , then  $\text{inv}(\sigma) \subseteq \text{inv}(\pi)$ , i.e., when the stack-sort operator is applied to  $\pi$ , each inversion in  $\pi$  either becomes a non-inversion in the output  $\sigma$ , or it stays an inversion. Non-inversions in  $\pi$  must also be non-inversions in  $\sigma$ .

The second step of the algorithm proceeds as follows. Suppose we have a candidate  $\lambda \in \text{cand}_d(p)$  and a permutation  $\pi$  containing  $\lambda$ . An inversion in  $\lambda$ , corresponding to the elements  $a$  and  $b$  in  $\pi$ , will remain an inversion after a pass through the stack if either of the two cases below applies.

- (1) There is an element  $c$ , larger than  $a$ , that appears between  $a$  and  $b$  in  $\pi$ . In this case  $a$  would be pushed onto the stack, and subsequently popped off by  $c$ , before  $b$  would be pushed onto the stack. The relative order of  $a$  and  $b$  would therefore remain the same, after applying the stack-sort operator. This corresponds to the pattern  $C_1$  in Tab. 2.
- (2) The stack is full of elements larger than  $a$ , when  $a$  appears in the input. In this case  $a$  bypasses the stack and appears before  $b$  in the output. This happens precisely when there is a sequence of decreasing elements, all larger than  $a$ , that appear in  $\pi$  before  $a$ . This sequence also has the property that all its elements are left-to-right maxima in  $\pi$ . Otherwise, at least one element of the sequence would be popped of the stack before  $a$  appears. For a stack of depth  $d+1$ , this property is described by the pattern

$$V_d = (d(d-1) \cdots 1, \{(i, j) \in \llbracket 0, d \rrbracket \times \llbracket 0, d \rrbracket : (d-i) > j\}).$$

This case corresponds to the pattern  $C_2$  in Tab. 2, where the region decorated with  $V_d$  must contain the pattern  $V_d$ .

If neither of the two cases applies to an inversion in  $\lambda$ , it must become a non-inversion in the output. This corresponds to the pattern  $C_3$  in Tab. 2, where the shaded region decorated with  $V_d$  must avoid the pattern  $V_d$ . This process is formalized in Algorithm 3, which extends [CU12, Algorithm 1].

By choosing  $p = 21$  and applying Algorithm 3 we easily arrive at the following proposition.

**Input:** The depth  $d$  of the stack; a pattern  $p$ ; and  $\lambda \in \text{cand}_d(p)$

**Output:** A (possibly empty) set of decorated patterns  $T$

```

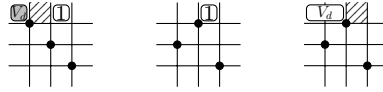
1 Let  $T = \{(\lambda, \emptyset, \emptyset, \emptyset, \emptyset)\}$  and let  $n$  be the length of  $\lambda$ 
2 for  $(i, j) \in \text{inv}(\lambda)$  do
3   Let  $R_1 = \llbracket \lambda^{-1}(i), \lambda^{-1}(j) - 1 \rrbracket \times \llbracket i, n \rrbracket$  and  $R_2 = \llbracket 0, \lambda^{-1}(i) - 1 \rrbracket \times \llbracket i, n \rrbracket$ 
4   for  $r = (\lambda, S, M, D, C) \in T$  do
5     Remove  $r$  from  $T$ 
6     if  $(i, j) \in \text{inv}(p)$  then
7       if  $R_1 \not\subseteq S$  then
8         if An element of  $\lambda$  is contained in  $R_1$  then Add  $r$  to  $T$ 
9         else Add  $(\lambda, S, M \cup \{(R_1 \setminus S, 1)\}, D, C)$  to  $T$ 
10      if No element of  $\lambda$ , no marking in  $M$ , or decoration in  $C$ , is contained in
11       $R_1 \cup S$  and  $R_2$  is not contained in any decoration in  $D$  then
12        Add  $(\lambda, S \cup R_1, M, D, C \cup \{(R_2, V_{d-1})\})$  to  $T$ 
13      else if No element of  $\lambda$ , no marking in  $M$ , or decoration in  $C$ , is contained
14      in  $R_1 \cup S$  and no decoration from  $C$  is contained in  $R_2$  then
15        Add  $(\lambda, S \cup R_1, M, D \cup \{(R_2, V_{d-1})\}, C)$  to  $T$ 

```

**Algorithm 3:** An algorithm describing how a particular candidate must be decorated

**Proposition 3.2.** *Stack-sortable permutations, with a stack of depth  $d$ , are precisely the avoiders of 231 and  $(d+1)d \cdots 21$ .*

Note that Goodrich et al. [GG<sup>+</sup>12] independently discovered this proposition. To describe the permutations sortable by two passes through a stack of depth  $d$ , one would need to describe the preimage of the avoiders of  $p = 231$ . In this case  $\text{cand}_d(p) = \{231, 321\}$  (if  $d > 1$ ). Algorithm 3 then gives



To complete the description, one also needs to determine the preimage of  $(d+1)d \cdots 1$ , see [MU12].

Description	Candidates	Output
Inversions that stay inverted	$C_1 = $ , $C_2 = $	
Inversions that become non-inversions	$C_3 = $	

TABLE 2. The decorations, markings and shadings needed for a stack of depth  $d$

**3.2. Sorting with a queue.** A *queue* is a list with the restrictions that elements can only be added to one end of the list, called the *front* and removed from the other end of the list, called the *back*. The act of adding an element to a queue is called *enqueueing*, and the act of removing is called *dequeuing*.

Given an input permutation  $\pi$ , let  $Q(\pi)$  be the permutation obtained by the following procedure [Knu75]: (1) If the queue is empty or the last element is smaller than the first element of the input,  $s$ , enqueue  $s$ . (2) Otherwise, dequeue elements from the queue into the output permutation until the front element of the queue is larger than  $s$ , or the queue is empty. Add  $s$  to the output permutation. (3) Repeat this process until the input is empty. (4) Empty the queue into the output permutation.

We will now sketch an algorithm for describing the preimage of any set defined by the avoidance of classical patterns. More precisely, given a classical pattern  $p$ , the algorithm outputs a set  $M$  of decorated patterns such that  $\text{Av}(M) = \{\pi : Q(\pi) \in \text{Av}(p)\}$ . In [MU12] we give a proposition analogous to Proposition 3.1, but here we consider the candidates  $\{\lambda : \text{inv}(p) \subseteq \text{inv}(\lambda)\}$  for simplicity. The preimage algorithm for  $Q$  proceeds similar to the algorithm for a stack of depth  $d$ , except when we consider how inversions change, we use the patterns shown in Tab. 3.

Candidates	Output
$D_1 = $ , $D_2 = $	
$D_3 = $	

TABLE 3. The decorations, markings and shadings needed for a queue

Let  $r$  denote the reverse of a permutation, and  $c$  denote the complement of a permutation. By using the algorithm for a queue and a stack [CU12], the following theorem is proved.

**Theorem 3.3.** *A permutation  $\pi$  avoids 4312 if and only if  $(S \circ r \circ c \circ Q)(\pi)$  is sorted.*

Since all the maps in the composition  $S \circ r \circ c \circ Q$  are linear time operators and it takes linear time to check whether the output is sorted, this provides a linear time algorithm for checking for the avoidance of 4312. Such linear time algorithms have only been known for patterns of length at most 3 as well as the increasing and decreasing patterns of any length ( $12 \cdots k$  and  $k \cdots 21$ ). Also see Albert et al. [AAAH01] for algorithms with running time  $n \log n$  for patterns of length 4.

One can similarly find preimage algorithms for a pop-stack, insertion sort and pancake sort, see [MU12]. In all of the algorithms above, we have only considered preimages of classical pattern classes. However, in [MU12], we extend the preimage algorithm for a stack to handle certain mesh pattern classes. This allows us to give an automatic proof of the description of the West-3-stack-sortable permutations.

## ACKNOWLEDGMENTS

The second author would like to thank Sara Billey for asking the question that led to the development of the BiSC algorithm as well as helpful discussions on how to implement it. He would also like to thank Anders Claesson for suggesting that the forbidden patterns might somehow be inferred from the allowed patterns. He would finally like to thank Einar Steingrímsson for many helpful discussions on permutation patterns. The name of the algorithm is derived from the names of these three people.

## REFERENCES

- [AAAH01] Michael H. Albert, Robert E. L. Aldred, Mike D. Atkinson, and Derek A. Holton. Algorithms for pattern involvement in permutations. In *Algorithms and computation (Christchurch, 2001)*, volume 2223 of *Lecture Notes in Comput. Sci.*, pages 355–366. Springer, Berlin, 2001.
- [AAB<sup>+</sup>11] M. H. Albert, M. D. Atkinson, M. Bouvel, A. Claesson, and M. Dukes. On the inverse image of pattern classes under bubble sort. *J. Comb.*, 2(2):231–243, 2011.
- [Atk98] M. D. Atkinson. Permutations which are the union of an increasing and a decreasing subsequence. *Electron. J. Combin.*, 5:Research paper 6, 13 pp. (electronic), 1998.
- [BC11] P. Brändén and A. Claesson. Mesh patterns and the expansion of permutation statistics as sums of permutation patterns. *Electron. J. Combin.*, 18, 2011.
- [Bil11] S. Billey. Consequences of the Lakshmibai-Sandhya theorem. AWM Anniversary Conference, 2011.
- [BMB07] M. Bousquet-Mélou and S. Butler. Forest-like permutations. *Ann. Comb.*, 11(3-4):335–354, 2007.
- [Cla12] A. Claesson. Personal communication, 2012.
- [CPW11] A. Crites, G. Panova, and G. S. Warrington. Shape and pattern containment of separable permutations, arxiv:1011.5491. 2011.
- [CU12] A. Claesson and H. Ulfarsson. Sorting and preimages of pattern classes. In *24th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2012)*, Discrete Math. Theor. Comput. Sci. Proc., AR, pages 595–606. Assoc. Discrete Math. Theor. Comput. Sci., Nancy, 2012.
- [DPTW12] M. Dairiko, L. Pudwell, S. Tyner, and C. Wynn. Non-contiguous pattern avoidance in binary trees. Special Session on Permutation Patterns, Algorithms and Enumerative Combinatorics, AMS Fall Eastern Sectional Meeting, 2012.
- [GG<sup>+</sup>12] T. Goodrich, D. Groth, , L. Knop, and L. Pudwell. Sorting permutations with a finite-depth stack. Indiana MAA Section Meeting, Ball State University, Muncie, Indiana, March 24, 2012.
- [Han12] S. Hannah. Personal communication, 2012.
- [Hoa62] C. A. R. Hoare. Quicksort. *Comput. J.*, 5:10–16, 1962.
- [Knu75] D. E. Knuth. *The art of computer programming*. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, second edition, 1975. Volume 1: Fundamental algorithms, Addison-Wesley Series in Computer Science and Information Processing.
- [LS90] V. Lakshmibai and B. Sandhya. Criterion for smoothness of Schubert varieties in  $Sl(n)/B$ . *Proc. Indian Acad. Sci. Math. Sci.*, 100(1):45–52, 1990.
- [MU12] H. Magnusson and H. Ulfarsson. Sorting operators and their preimages. In preparation, 2012.
- [Slo12] N. J. A. Sloane. The on-line encyclopedia of integer sequences, published electronically at <http://www.oeis.org>, 2012.
- [Ulf11] H. Ulfarsson. A unification of permutation patterns related to Schubert varieties. *Pure Math. Appl. (P.U.M.A.)*, 22(2):273–296, 2011.
- [Ulf12a] H. Ulfarsson. BiSC: An algorithm for discovering generalized permutation patterns. In preparation, 2012.

- [Ulf12b] H. Ulfarsson. Describing West-3-stack-sortable permutations with permutation patterns. *Electron. J. Combin.*, 67:Article B67d, 20 pp. (electronic), 2012.
- [Wes90] J. West. *Permutations with forbidden subsequences and stack-sortable permutations*. PhD thesis, MIT, 1990.

SCHOOL OF COMPUTER SCIENCE, REYKJAVÍK UNIVERSITY, MENNTAVEGI 1, 101 REYKJAVÍK, ICELAND

*E-mail address:* hjaltim07@ru.is, henningu@ru.is