SORTING OPERATORS AND THEIR PREIMAGES

Hjalti Magnússon
Master of Science
Computer Science
May 2013
School of Computer Science
Reykjavík University

M.Sc. RESEARCH THESIS

ISSN 1670-8539

Sorting operators and their preimages

by

Hjalti Magnússon

Research thesis submitted to the School of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science in **Computer Science**

May 2013

Research Thesis Committee:

Dr. Henning Arnór Úlfarsson, Supervisor
Assistant Professor, Reykjavík University

Dr. Michael Albert
Professor, University of Otago

Dr. Anders Claesson
Senior Lecturer, University of Strathclyde

Sorting operators and their preimages

Hjalti Magnússon

May 2013

Abstract

This thesis extends previous work of Claesson and Úlfarsson (2012) on algorithms for computing the preimage of a pattern class under the stack-sort operator. We consider several other sorting operators, namely stacks of fixed depth, queues, pop-stacks, insertion sort and pancake sort, and find corresponding algorithms for them. We also introduce the notion of meta patterns that allow us to present these algorithms in a uniform way. Furthermore, we consider how Claesson's and Úlfarsson's original algorithm for stack-sort can be extended to find the preimage of a mesh pattern class. This enables us to give an automatic proof of the description of West-3-stack-sortable permutations. Finally we show how the combination of the stack-sort and queue-sort operators can be used to create a linear time algorithm for determining avoidance of the pattern 4312.

Röðurnaraðferðir og formyndir þeirra

Hjalti Magnússon

Maí 2013

Útdráttur

Þessi ritgerð byggir á þeirri vinnu Claessons og Úlfarssonar sem snýr að reikniritum sem finna formyndir mynsturflokka undir staflaröðunarvirkjanum. Við skoðum einnig aðrar röðunaraðferðir, þ.e. röðun með stafla af takmarkaðri dýpt, röðun með biðröð, röðun með togstafla, innsetningarröðun og pönnukökuröðun, og gefum sambærileg reiknirit fyrir þær. Við kynnum einnig til sögunnar yfirmynstur sem gera okkur kleift að samræma framsetningu reikniritanna. Við sýnum jafnframt hvernig reiknirit Claesson og Úlfarssonar, fyrir formyndir staflaröðunar, er hægt að útfæra til þess að finna formyndir ákveðins flokks möskvamynstra. Þannig má sjálfvirknivæða sönnun á lýsingu þeirra umraðana sem hægt er að raða með þremur ítrunum af staflaröðun. Að lokum sýnum við hvernig samskeyting á staflaröðunar- og biðrararöðunarvirkjunum gefur reiknirit sem ákvarðar, á línulegum tíma, hvort umröðun innihaldi klassíska mynstrið 4312.

To Rapunzel. Sie machen Bio aus Liebe.

Acknowledgements

I would like to thank my advisor, Henning Úlfarsson, for his time, support, guidance and for providing me with a very interesting research topic.

I would like to thank Björn Þór Jónsson and Sigrún Ammendrup for their support throughout my studies at Reykjavik University.

I would like to thank Bjarki Ágúst Guðmundsson for proofreading my thesis and providing me with useful comments and suggestions.

I would like to thank Anders Claesson and Michael Albert for useful input and suggestions.

I would like to thank Vignir Örn Guðmundsson for introducing me to alt-J.

Finally, I would like to thank Einar Steingrímsson, whose enthusiasm sparked my interest in mathematics and whose vision gave me the chance to study it properly.

This work was partially supported by grant no. 090038014–5 from the Icelandic Research Fund.

Contents

List of Figures **xi**

List of Tables **xii**

List of Algorithms **xiii**

1 Introduction **1**
1.1 Permutations 2
1.2 Permutation patterns 3
1.2.1 Decorated patterns 3
1.3 Sorting operators and preimages 9
1.3.1 Sorting with a stack 10

2 Preimages of sorting operators **13**
2.1 Sorting with a stack of any depth 13
2.1.1 Meta patterns 16
2.2 Sorting with a queue 21
2.3 Sorting with a pop-stack 26
2.4 Insertion sort 29
2.5 Pancake sort 32
2.5.1 More expressive meta patterns 34
2.6 A linear-time algorithm for pattern avoidance 43

3 Preimages of mesh patterns **45**
3.1 Finding preimages of mesh patterns 47
3.2 West-3-stack-sortable permutations 53

4 Conclusions **57**

A Implementation of preimage algorithms **61**

A.1 Implementation in Sage 61
A.2 Sorting with a stack of depth d 63

B Meta patterns for $\mathsf{meshpreim}_S$ **65**

List of Figures

1.1 Stack-sort operator applied to 41325 11

2.1 Stack-sort operator of depth 3 applied to 45321 14
2.2 Stack-sort operator of depth 4 applied to 45321 14
2.3 Queue-sort operator applied to 235416 22
2.4 Pop-stack-sort operator applied to 51243 27
2.5 Pancake-sort operator applied to 214356 33

3.1 The meta patterns describing the domain of $\mathsf{meshpreim}_S$ 48

List of Tables

2.1 Meta patterns for a stack of depth d 18
2.2 Meta patterns for a queue 23
2.3 Meta patterns for a pop-stack 28
2.4 Meta patterns for insertion sort 31
2.5 Meta patterns for pancake sort 39
2.6 Meta patterns for pancake sort 40

3.1 The patterns defining the preimage of w_2 under S 55

B.1 Meta patterns for $\mathsf{meshpreim}_S$ – Case 1 65
B.2 Meta patterns for $\mathsf{meshpreim}_S$ – Case 2 65
B.3 Meta patterns for $\mathsf{meshpreim}_S$ – Case 3 66
B.4 Meta patterns for $\mathsf{meshpreim}_S$ – Case 4 66
B.5 Meta patterns for $\mathsf{meshpreim}_S$ – Case 5 66
B.6 Meta patterns for $\mathsf{meshpreim}_S$ – Case 6 67
B.7 Meta patterns for $\mathsf{meshpreim}_S$ – Case 7 67
B.8 Meta patterns for $\mathsf{meshpreim}_S$ – Case 8 68
B.9 Meta patterns for $\mathsf{meshpreim}_S$ – Case 9 69
B.10 Meta patterns for $\mathsf{meshpreim}_S$ – Case 10 70

List of Algorithms

2.1 decorate$_{S_d}$ 19
2.2 decorate$_{Q}$ 24
2.3 decorate$_{T}$ 28
2.4 decorate$_{I}$ 31
2.5 decorate$_{P}$ 41

3.1 decorate$_{S}$ 46

A.1 decorate$_{S_d}$ (without meta patterns) 63

Chapter 1

Introduction

Although the notion of permutation patterns is implicit in the literature a long way back, the formal study of permutation patterns began in the 1960's, inspired by the stack-sorting problem introduced by Knuth [1, Ch. 2.2.1, pp. 242–243]. Knuth discovered that permutations that could be generated by passing the permutation $12\dots n$ through a stack (or equivalently, the permutations sortable by a pass through a stack) are precisely the permutations that do not contain a subsequence a, b, c, such that $c < a < b$. In terms of permutation patterns, Knuth discovered that the permutations, sortable by a single pass through a stack, are precisely the class of permutations defined by the avoidance of the pattern 231. The problem of describing the permutations sortable by two passes through a stack then remained open, until in 1990 when West [2] gave their description by extending the notion of classical permutation patterns. In 2012 Úlfarsson [3] gave a description of permutations sortable by three passes through a stack by extending a class of permutation patterns called mesh patterns, defined by Brändén and Claesson [4].

In [5] Claesson and Úlfarsson gave an algorithm for describing the preimage of any class of permutations, defined by the avoidance of a classical pattern, under the stack-sort operator. This algorithm generalizes the results of Knuth, which gave a description of the preimage of permutations avoiding 21, and the results of West, which described the preimage of the permutations avoiding 231.

This thesis extends the work of Claesson and Úlfarsson in two ways. First, we give algorithms that describe the preimage of five sorting operators, namely stack sort of limited depth, queue sort, pop-stack sort, insertion sort and pancake sort (Chapter 2). These algorithms are based on the preimage algorithm for stack sort, however, we also introduce the notion of meta patterns, which allow us to give a uniform description of these algorithms. We also show how these preimage algorithms can be applied to prove the correctness

of a linear-time algorithm for determining the avoidance of the pattern 4312. We also present an algorithm to give a description of the preimage of pattern classes, described by the avoidance of a certain type of mesh patterns, under the stack-sort operator (Section 3). This allows us to describe the West-3-stack sortable permutations. Finally, we give an implementation of these algorithms in the computer algebra system Sage (Appendix A).

1.1 Permutations

Let $[\![a, b]\!]$ denote the integer interval $\{i \in \mathbb{Z} : a \leq i \leq b\}$. A *permutation of rank* n is a bijection $\pi : [\![1, n]\!] \to [\![1, n]\!]$. The set of all permutations of rank n is denoted by $\mathfrak{S}_n$.

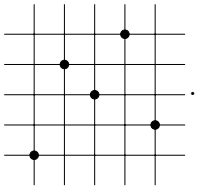
A permutation $\pi \in \mathfrak{S}_n$ can be represented in *one-line notation* as the word

$$\pi = \pi(1)\pi(2)\dots\pi(n).$$

The *graph* of a permutation $\pi \in \mathfrak{S}_n$ is defined as

$$G(\pi) = \{(i, \pi(i)) : i \in [\![1, n]\!]\}$$

and is depicted by plotting the points of $G(\pi)$ on a grid. For example, the graph of 14352 is depicted as



We let π_i^v denote the i-th vertical line of (the graph of) π and π_i^h denote the i-th horizontal line of π. Furthermore, if $\pi \in \mathfrak{S}_n$, we let π_0^h, π_{n+1}^h, π_0^v and π_{n+1}^v denote the bottom, top, left and right edge of π, respectively.

Let $\pi = \pi_1\pi_2\dots\pi_n \in \mathfrak{S}_n$. We define

$$\pi^r = \pi_n\pi_{n-1}\dots\pi_1$$

as the *reverse* of π,

$$\pi^c = ((n+1) - \pi_1)((n+1) - \pi_2) \dots ((n+1) - \pi_n)$$

as the *complement* of π and π^i as the (usual functional) *inverse* of π.

An *inversion* in π is a pair (π_i, π_j) where $\pi_i > \pi_j$ and $i < j$. If a pair (π_i, π_j) is not an inversion, we refer to it as a *non-inversion*. We let $\mathsf{inv}(\pi)$ denote the set of all inversions in π and $\mathsf{ninv}(\pi)$ denote the set of all non-inversions in π.

A *descent* in π is an index i, such that $\pi_i > \pi_{i+1}$. In this case π_{i+1} is called the *descent bottom*.

We also define $\mathsf{id}_n = 123 \dots n$ as the *identity permutation of rank* n, and ε as the *empty permutation*, which is the unique bijection $\varnothing \to \varnothing$.

1.2 Permutation patterns

Let $\sigma \in \mathfrak{S}_m$ and $\pi \in \mathfrak{S}_n$ be permutations. We say that π *contains* σ (or σ *occurs* in π) if there exists a subsequence in π with the same relative order as σ. In this context, the permutation σ is called a *(classical) pattern*. If π does not contain σ we say that π *avoids* σ.

For example, if $\pi = 41523$ we have that π contains the pattern 231 since the subsequence 452 ($\underline{41}\underline{52}3$) in π has the same relative order as 231. Furthermore, we have that π avoids the pattern 321, since no subsequence of π has that relative order.

1.2.1 Decorated patterns

Let $\mathbb{N} = \{0, 1, \dots\}$ and let $\mathbb{N}_+ = \mathbb{N} \setminus \{0\}$. A *decorated pattern* is a 5-tuple $p = (\pi, S, M, A, C)$ where

- $\pi \in \mathfrak{S}_n$ is the *underlying classical pattern* of p,
- $S \subseteq [\![0, n]\!] \times [\![0, n]\!]$ are the *shadings* of p,
- $M \subseteq \{(T, k) : T \subseteq [\![0, n]\!] \times [\![0, n]\!], k \in \mathbb{N}_+\}$ are the *markings* of p,
- $A \subseteq \{(T, q) : T \subseteq [\![0, n]\!] \times [\![0, n]\!], q \text{ is a decorated pattern}\}$ are the *avoidance decorations* of p, and

- $C \subseteq \{(T, q) : T \subseteq [\![0, n]\!] \times [\![0, n]\!], q \text{ is a decorated pattern}\}$ are the *containment decorations* of p.

Decorated patterns were introduced by Úlfarsson in [3]. If $M = A = C = \varnothing$, we refer to p as a *mesh pattern*, which were introduced by Brändén and Claesson in [4]. If $A = C = \varnothing$, we refer to p as a *marked mesh pattern*, introduced by Úlfarsson in [6].

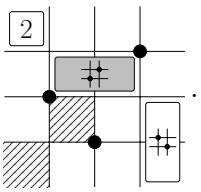
We depict a decorated pattern (π, S, M, A, C) with the graph of π, with the following additions.

- For each $(i, j) \in S$ we shade the unit square with bottom left corner (i, j) with diagonal lines.
- For each $(T, n) \in M$ we draw a box around the region T and mark it with n.
- For each $(T, q) \in A$ we draw a shaded box around the region T and mark it with the pattern q.
- For each $(T, q) \in C$ we draw a (non-shaded) box around the region T and mark it with the pattern q.

As an example, the pattern

$$(213, \{(0, 0), (1, 1)\}, \{(\{(0, 3)\}, 2)\}, \{(\{(1, 2), (2, 2)\}, 12)\}, \{(\{(3, 0), (3, 1)\}, 21)\})$$

is depicted as



We say that a decorated pattern (π, S, M, A, C) *occurs* in a permutation σ, if we can find the underlying classical pattern π in the graph of σ positioned in such a way that

- the shaded regions are not occupied by any elements of σ,
- the marked regions are occupied by at least the number of elements specified by the marking,
- the elements occupying the avoidance decorated regions avoid the pattern marked in the region,
- the elements occupying the containment decorated regions contain the pattern marked in the region.

More formally, an occurrence of $p = (\pi, S, M, A, C)$ in σ, where $\pi \in \mathfrak{S}_k$ and $\sigma \in \mathfrak{S}_n$, is a subset ω of the graph of σ, $G(\sigma)$, such that there are two injections $\alpha, \beta : [\![1, k]\!] \to [\![1, n]\!]$ that satisfy the following six conditions.

1. For all i, $\alpha(i) < \alpha(i+1)$ and $\beta(i) < \beta(i+1)$.
2. $\omega = \{(\alpha(i), \beta(j)) : (i, j) \in G(\pi)\}$.

Let

$$T_{ij} = [\![\alpha(i) + 1, \alpha(i+1) - 1]\!] \times [\![\beta(j) + 1, \beta(j+1) - 1]\!],$$

where we define $\alpha(0) = \beta(0) = 0$ and $\alpha(k+1) = \beta(k+1) = n+1$. This set contains all the points in (the graph of) the permutation σ after the pattern p has been "stretched" over σ to match the underlying classical pattern π. Furthermore, suppose $T \subseteq [\![0, n]\!] \times [\![0, n]\!]$. We then define

$$U(T) = \{(\alpha(i), \beta(j)) : (i, j) \in G(\pi) \text{ and } (i - t, j - s) \in T \text{ for all } t, s \in \{0, 1\}\}$$

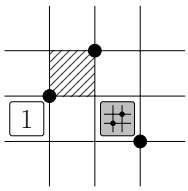
as the set of points surrounded by the region defined by T. We also define

$$V(T) = \left(\bigcup_{(i,j) \in T} T_{ij} \cap G(\sigma) \right) \cup U(T).$$

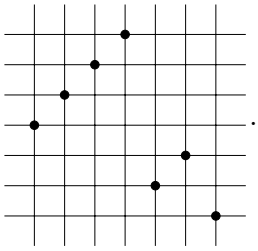
This set contains the points of the permutation σ that land in the region T, after p has been stretched over σ, along with the points in σ that correspond to points in the underlying classical pattern π that are surrounded by the region T in p. The remaining conditions are then as follows.

3. For all $(i, j) \in S$, we have $T_{ij} \cap G(\sigma) = \varnothing$.
4. For all $(T, n) \in M$, we have $|V(T)| \geq n$.
5. For all $(T, q) \in A$, we have that $V(T)$ avoids q.
6. For all $(T, q) \in C$, we have that $V(T)$ contains q.

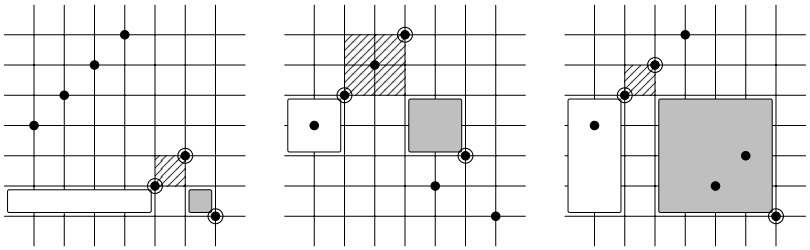
Example 1.2.1. Suppose we have the decorated pattern

$p =$ 

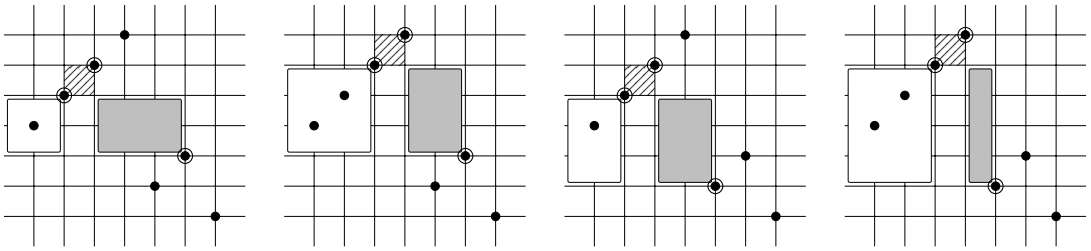
and the permutation

$\sigma =$  .

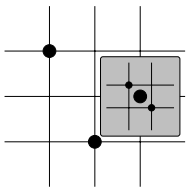
Below we show three examples where σ contains the underlying classical pattern of p, but not p. We denote the occurrence of the underlying pattern by circling the elements of σ. In the first example, the marking contains no points and is therefore not satisfied. In the second example the element 6 lands in a shaded region and in the third example, the avoidance decoration contains an occurrence of the pattern 12.



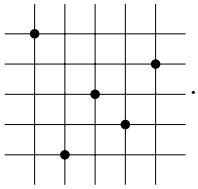
The pattern p, however, has four occurrences in σ, namely the following.



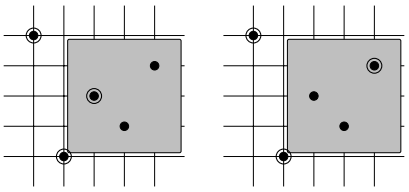
Example 1.2.2. Let

$p =$ 

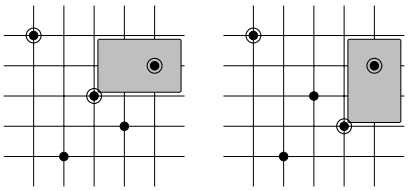
and

$\sigma =$  .

Below are two examples where σ contains the underlying classical pattern of p, but not p. In both cases the shaded region contains the pattern 21, which is forbidden by the avoidance decoration.



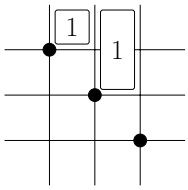
There are, on the other hand, two occurrences of p in σ, namely the following. In these two cases, the shaded region only contains the element corresponding to 2 in p, and therefore does not contain 21.



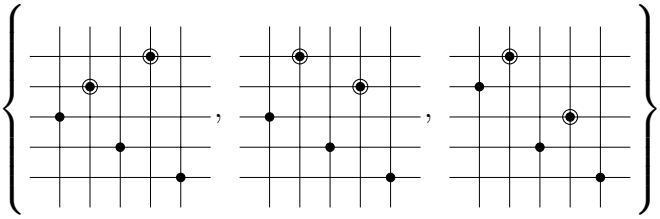
We let $\mathsf{Av}(M)$ denote the set (or *class*) of all permutations avoiding a set M of decorated patterns, and $\mathsf{Co}(M)$ the set of all permutations containing some pattern in M.

Remark. Markings and containment decorations, marked with classical patterns, do not add to the expressiveness of decorated patterns. Given a decorated pattern p, we can define a finite set $\mathsf{exp}(p)$ of decorated patterns, with no markings or containment decorations, with the property that a permutation π avoids p if and only if $p \in \mathsf{Av}(\mathsf{exp}(p))$. We call this set $\mathsf{exp}(p)$ the *expansion* of p.

For example, if

$p =$ 

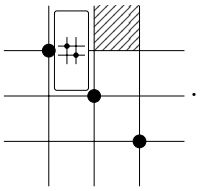
then we have

$\mathsf{exp}(p) =$ 

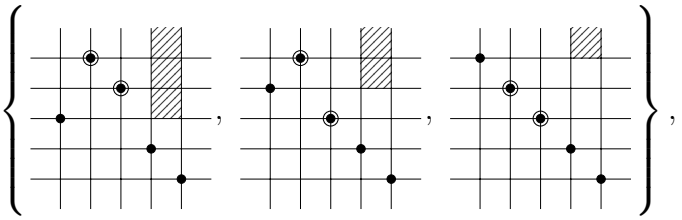
We obtain $\mathsf{exp}(p)$ by inserting points into the markings of p. There is only one way of inserting a point in the left marking. However, after we have inserted a point in the left marking, the right marking covers three boxes. Thus there are three distinct ways of

inserting a point into that region, so we obtain three patterns. The inserted points are circled in the set above.

As another example, suppose we have

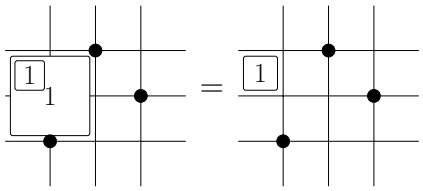
$q =$  .

The expansion of q,

$\mathsf{exp}(q) =$  ,

is then obtained by inserting an inversion in the region marked with the containment decoration. There are three distinct ways of inserting an inversion in the region, so we obtain three patterns.

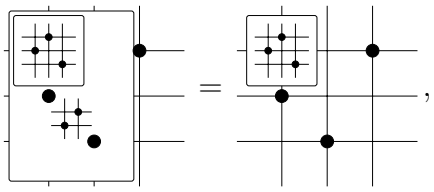
Remark. In some cases markings and decorations are redundant. For example, we have that



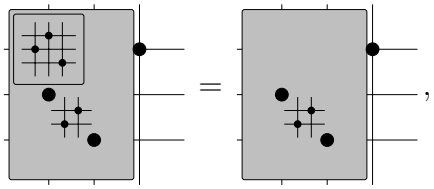
since by satisfying the smaller marking, we also satisfy the larger marking, making the larger marking redundant. In general, by the same argument, if we have a region T marked with k and another region T' marked with k', such that $T' \subseteq T$ and $k \leq k'$, we can remove the marked region T.

Furthermore if C and C' are containment decorations, decorated with the classical patterns p and p', respectively, such that C contains C' and p' (as a permutation) contains the pattern p, we can remove the decoration C. The decoration C is redundant, since if the decoration C' is satisfied, C is automatically satisfied as well. Similarly, if C and C' are avoidance decorations, decorated with the classical patterns p and p', respectively, such that C' contains C and p' (as a permutation) contains the pattern p, we can remove the

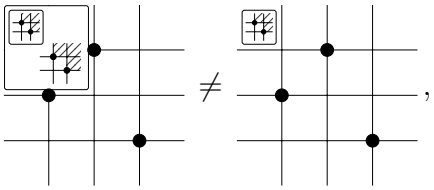
decoration C. Thus, for example, we have that



and



Note, however, that the same does not apply if decorations contain non-classical patterns. For example, we have that



since the pattern on the left hand side is avoided by 542631 whereas the pattern on the right hand side is not.

1.3 Sorting operators and preimages

Sorting, i.e., the rearrangement of items into ascending order, is one of the most extensively researched topics in computer science. A function $f : \mathfrak{S}_n \to \mathfrak{S}_n$ is called a *sorting operator* if there exists another function, $u : \mathbb{N} \to \mathbb{N}$, such that $f^{u(n)}(\pi) = \mathsf{id}_n$, for all $\pi \in \mathfrak{S}_n$.

Our main emphasis in this thesis will be finding preimages of classes of permutation under several sorting operators. More precisely, given a sorting operator f and a pattern p, we give algorithms to construct a set M of decorated patterns such that

$$\mathsf{Av}(M) = \{\pi : f(\pi) \in \mathsf{Av}(p)\},$$

or equivalently $f^{-1}(\mathsf{Av}(p)) = \mathsf{Av}(M)$. In this context we refer to p as the *target* pattern. The algorithms can then be extended to find the preimage of a class of permutations defined by the avoidance of a set of classical patterns L, which is given by

$$f^{-1}(\mathsf{Av}(L)) = \bigcap_{q \in L} f^{-1}(\mathsf{Av}(q)).$$

1.3.1 Sorting with a stack

A *stack (of unlimited depth)* 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*.

Given an input permutation π, let $S(\pi)$ be the permutation obtained by the following procedure.

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 and append them to the output permutation, until s can be pushed onto the stack.
3. Repeat this process until the input is empty.
4. Pop all elements off the stack and append them to the output permutation.

We refer to S as the *stack-sort operator*. For a permutation $\pi \in \mathfrak{S}_n$, if $S(\pi) = \mathsf{id}_n$, we say that π is *stack-sortable*.

The stack-sort operator can also be defined recursively as follows [7]. For a permutation $\pi = \alpha n \beta$, where n is the largest element of π, we have that

$$\begin{aligned} S(\pi) &= S(\alpha n \beta) = S(\alpha) S(\beta) n, \\ S(\varepsilon) &= \varepsilon. \end{aligned}$$

Figure 1.1 shows the result of applying the stack-sort operator to the permutation 41325. First 4 and 1 are pushed onto the stack. The 1 is then popped off the stack, by the 3, which is pushed onto the stack. The 2 is then pushed onto the stack, on top of the 3. Finally, since 5 is larger than all the elements on the stack, all the elements are popped off the stack, and the 5 is pushed onto the stack. The stack is then emptied into the output.

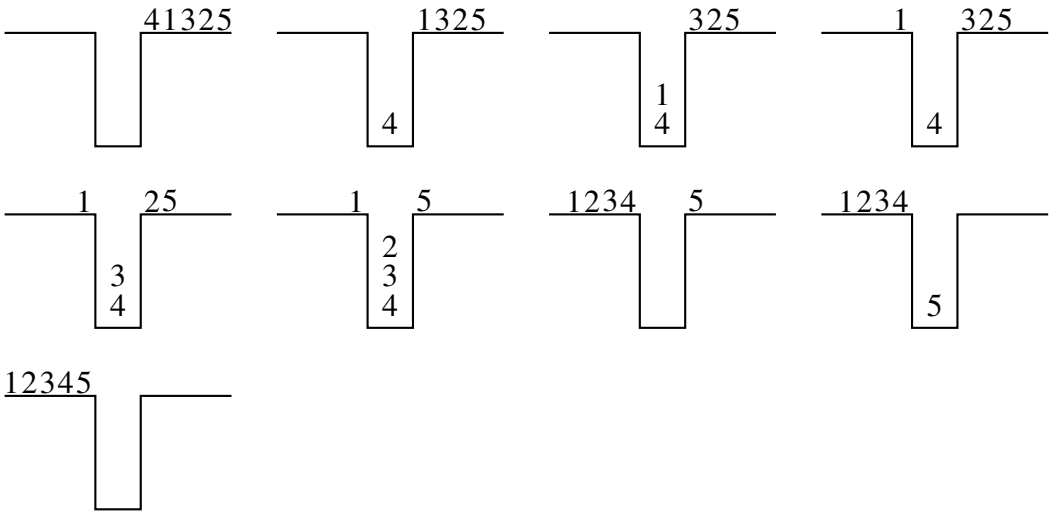


Figure 1.1: Stack-sort operator applied to 41325

As mentioned before, Knuth discovered that the stack-sortable permutations were precisely the permutations that avoided the pattern 231. Since the identity permutation is the only permutation that avoids the pattern 21, we can formulate Knuth's discovery as $S^{-1}(\mathsf{Av}(21)) = \mathsf{Av}(231)$. Using this result, we have that if a permutation avoids 231 after a pass through a stack, it is sortable by two passes through a stack. The discovery of West, which originally described the permutations sortable by two passes through a stack, can therefore be written as

$$S^{-1}(\mathsf{Av}(231)) = \mathsf{Av}\left(\quad,\quad\right).$$

Chapter 2

Preimages of sorting operators

2.1 Sorting with a stack of any depth

The *depth* of a stack is the number of elements it can hold. The stack-sort operator can be extended as follows. Given a stack of depth d and an input permutation π, let $S_d(\pi)$ be the permutation obtained by the following procedure.

1. If the stack is empty, push the first element of the input onto the stack.
2. If the stack is full (i.e., contains d elements), pop the top element from the stack and append it to the output permutation.
3. If the topmost element is larger than the first element of the input, s, push s onto the stack.
4. Otherwise, pop elements from the stack and append them to the output permutation, until s can be pushed onto the stack.
5. Repeat this process until the input is empty.
6. Pop all elements off the stack and append them to the output permutation.

We refer to S_d as the *stack-sort operator of depth* d. For a permutation $\pi \in \mathfrak{S}_n$, if $S_d(\pi) = \mathsf{id}_n$, we say that π is *stack-sortable with a stack of depth* d.

Figure 2.1 shows the result of applying the stack-sort operator of depth 3 to the permutation 45321. First 4 is pushed onto the stack, and immediately popped by the 5. The 3 and the 2 are then pushed onto the stack, and the stack becomes full. The 2 is therefore immediately popped off the stack. The same applies to the 1, which is pushed onto the stack and then immediately popped off the full stack. The stack is subsequently emptied

into the output. Figure 2.2 shows the result of applying the stack-sort operator of depth 4 to the same permutation. In this case, the stack is not filled by the 2, and the 1 can therefore be pushed onto the stack, on top of the 2. The relative order of 1 and 2 is therefore reversed in the output.

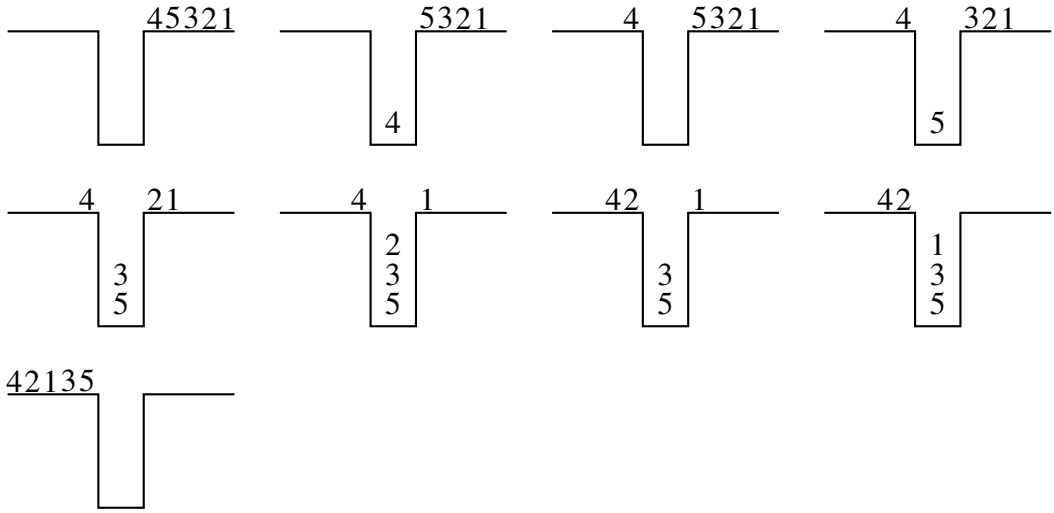


Figure 2.1: Stack-sort operator of depth 3 applied to 45321

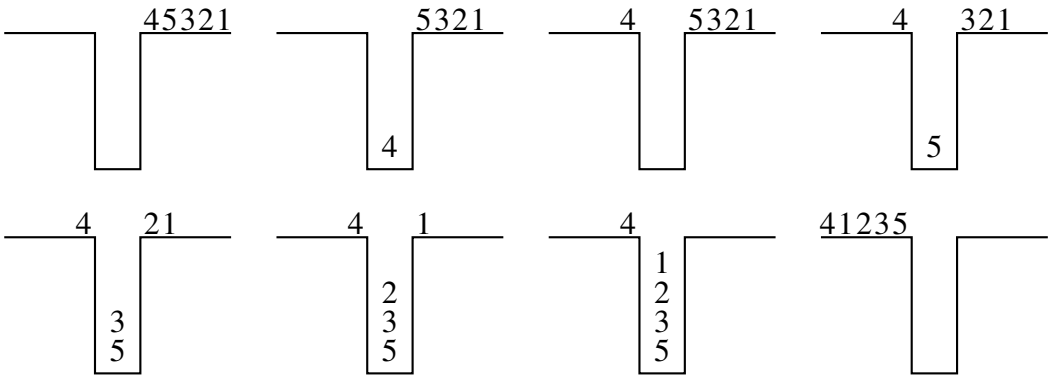


Figure 2.2: Stack-sort operator of depth 4 applied to 45321

The following definition is analogous to the recursive definition of the stack-sort operator. For a permutation $\pi = \alpha n \beta$, where n is the largest element of π, we have that

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

$$S_d(\varepsilon) = \varepsilon.$$

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. [8]).

In [5, Algorithm 1], Claesson and Úlfarsson presented an algorithm that gives a description of the preimage of any set, defined by the avoidance of classical patterns, for S. The algorithm takes as input the pattern defining the avoidance class whose preimage we want to find, and then 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, that finds the preimage of a stack of any depth. More precisely, given a classical pattern p, the generalized algorithm, preim_{S_d}, gives a set of decorated patterns that describe the preimage of $\mathsf{Av}(p)$ under S_d.

The analog of [5, Proposition 4.1], which generates classical pattern candidates, is the following.

Proposition 2.1.1. *Let $p = \alpha n \beta$ be a permutation of a finite set of integers where n is the largest element of p and $\alpha = a_1 a_2 \dots a_i$. Then*

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

contains all classical patterns that can become p after one pass through a stack of depth d.

Proof. In the case when $d = 1$, we have that S_d is the identity map, and therefore the only classical pattern that can become p, after S_d has been applied to it, is p itself. Furthermore, the only permutation that can become the empty permutation, ε, after a pass through a stack of any depth is ε itself.

Suppose then we have a pattern $p = \alpha n \beta$, and suppose we have a pattern $q = \gamma n \delta$ that becomes p after a pass through a stack of depth d. We know that, since n is the largest element of q, it cannot move to the left, and must therefore move to the right or stay in the same place after S_d is applied. Furthermore, since n is the largest element of q, we have that all the elements to the left of n in q must be to the left of n in p, since n pops all elements, smaller than itself, off the stack. Finally, we note that after n is pushed onto the stack, no element in q will pop n off the stack, and therefore the stack's depth is decreased

by (at least) 1. By the induction hypothesis, we therefore have that

$$\bigcup_{j=0}^{i} \{\gamma n \delta : \gamma \in \mathsf{cand}_d(a_1 a_2 \dots a_j), \delta \in \mathsf{cand}_{d-1}(a_{j+1} \dots a_i \beta)\}$$

contains all patterns that can become p after a pass of stack-sort of depth d. □

We note that if $\sigma = S_d(\pi)$, then $\mathsf{inv}(\sigma) \subseteq \mathsf{inv}(\pi)$. This follows from the fact that when the stack-sort operator is applied to π, each inversion in π either becomes a non-inversion in the output σ, or it stays an inversion. Non-inversions in π must also be non-inversions in σ.

In the second step of preim_{S_d} we consider the inversions of each candidate, and compare them to the inversions of the target pattern. In order for an inversion in the candidate to remain an inversion in the target, after a pass through a stack, in general we must add restrictions to the candidate. The same applies for inversions that become non-inversions. In order to generalize the process of adding restrictions to patterns we present the notion of a meta pattern.

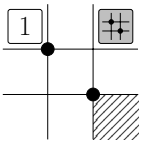
2.1.1 Meta patterns

A *meta pattern* is a decorated pattern which we use to add restrictions to decorated patterns. Given a meta pattern $\mathcal{P}$, and a decorated pattern p, we can *apply* $\mathcal{P}$ to a subsequence σ of the underlying pattern of p by stretching $\mathcal{P}$ over p, so that the underlying pattern of $\mathcal{P}$ matches σ. We then add the restrictions from $\mathcal{P}$ to p. In order for $\mathcal{P}$ to be applicable to σ, the following conditions must be met.

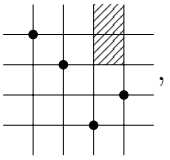
1. The subsequence σ has the same relative order as the underlying pattern of $\mathcal{P}$.
2. The added restrictions from $\mathcal{P}$ must not conflict with the restrictions of p:
   (a) No element of p can be contained in a shaded region of $\mathcal{P}$.
   (b) A containment decoration or marking cannot be contained in a shaded region.
   (c) A containment decoration, marked with a classical pattern q, cannot be contained in an avoidance decoration, marked with a classical pattern r, if containment of q implies containment of r.

(We will introduce a more expressive type of meta patterns in Section 2.5.)

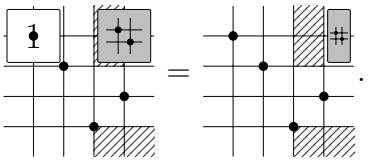
For example, if

$\mathcal{P} =$ 

and

$p =$  ,

we can apply $\mathcal{P}$ to 31 in p and obtain

 .

Note that we can remove the marking after applying $\mathcal{P}$, since it contains an element from p. Furthermore, since a shading is more restrictive than an avoidance decoration, we can contract the avoidance decoration from $\mathcal{P}$ in p. It is not possible to apply $\mathcal{P}$ to 43 in p, since in that case the elements 1 and 2 in p would land inside the shading from $\mathcal{P}$.

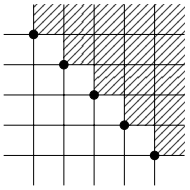
With the notion of meta patterns, we can now describe $\mathsf{decorate}_{S_d}$, the second step of preim_{S_d}. Suppose we have a candidate $\lambda \in \mathsf{cand}_d(p)$ and a permutation π containing λ. An inversion in λ, corresponding to the elements a and b in π, will remain an inversion after a pass through a stack of depth d if either of the two cases below applies.

1. There is an element c, larger than a, that appears between a and b in π. In this case a is pushed onto the stack, and subsequently popped off by c, before b is pushed onto the stack. The relative order of a and b therefore remains the same after applying the stack-sort operator. This corresponds to the meta pattern $\mathcal{C}_1$ in Table 2.1.

2. The stack contains $d-1$ elements, all larger than a, when a appears in the input. In this case a is pushed onto the stack, and immediately popped off, since the stack becomes full, and thus a appears before b in the output. This happens precisely when there is a sequence of $d-1$ decreasing elements, all larger than a, that appear in π before a. This sequence also has the property that all its elements are right-to-left maxima in π. Otherwise, at least one element of the sequence would be popped off the stack before a appears. This property is described by the pattern

$$V_d = ((d-1)(d-2)\dots 1, \{(i,j) \in [\![0,d-1]\!] \times [\![0,d-1]\!] : (d-1-i) < j\}) .$$

As an example

$V_6 =$  .

This case corresponds to the meta pattern $\mathcal{C}_2$ in Table 2.1. Note that the shading is not necessary, but is added to exclude the first case.

If neither of the two cases applies to an inversion in λ, it must become a non-inversion in the output. This corresponds to the meta pattern $\mathcal{C}_3$ in Table 2.1.

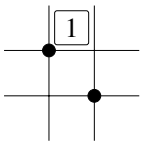
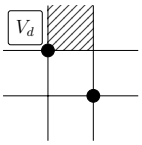
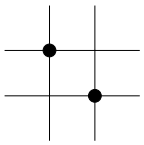
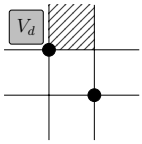
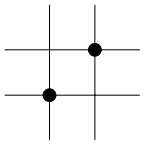
Description	**Candidates**	**Target**
Inversions that stay inverted	$\mathcal{C}_1 =$  , $\mathcal{C}_2 =$ 	
Inversions that become non-inversions	$\mathcal{C}_3 =$ 	

Table 2.1: Meta patterns applied to inversions in candidates to obtain inversions or non-inversions in the target after a single pass through a stack of depth d

Algorithm 2.1 gives a formal description of $\mathsf{decorate}_{S_d}$. For comparison, Algorithm A.1, which extends [5, Algorithm 1] and is written in a similar style, gives a description of $\mathsf{decorate}_{S_d}$ without the use of meta patterns.

We then define

$$\mathsf{preim}_{S_d}(p) = \bigcup_{\lambda \in \mathsf{cand}_d(p)} \mathsf{decorate}_{S_d}(d, p, \lambda),$$

and present the following theorem.

Theorem 2.1.2. *The preimage of the pattern class, defined by the avoidance of p, under the stack-sort operator of depth d is*

$$S_d^{-1}(\mathsf{Av}(p)) = \mathsf{Av}\left(\mathsf{preim}_{S_d}(p)\right).$$

Proof. This statement holds since $\mathsf{cand}_d(p)$ contains all classical patterns that can become p after a pass through a stack of depth d. In $\mathsf{decorate}_{S_d}$ restrictions are then added to the candidates, such that any occurrence of a restricted candidate in a permutation must

Algorithm 2.1: $\mathsf{decorate}_{S_d}$

Input: The depth d of the stack; a target pattern p; and $\lambda \in \mathsf{cand}_d(p)$
Output: A (possibly empty) set of decorated patterns M

```
Let M = {(λ, ∅, ∅, ∅, ∅)};
for v ∈ inv(λ) do
    for r ∈ M do
        Remove r from M;
        if v ∈ inv(p) then
            for i ∈ {1, 2} do
                if C_i is applicable to v in r then apply C_i to v in r and add to M;
        else
            if C_3 is applicable to v in r then apply C_3 to v in r and add to M;
return M;
```

become an occurrence of p after the permutation has passed through a stack of depth d. □

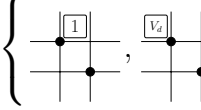
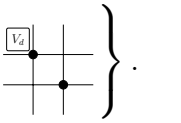
Remark. Let $\mathcal{C}_2'$ be the meta pattern $\mathcal{C}_2$ without the shading. As noted above, the shading in $\mathcal{C}_2$ is not necessary, and is only added to exclude the cases to which $\mathcal{C}_1$ applies. Thus, we can substitute $\mathcal{C}_2$ for $\mathcal{C}_2'$ in $\mathsf{decorate}_{S_d}$. However, for an inversion in a candidate λ, we need only apply $\mathcal{C}_2'$ if $\mathcal{C}_2$ is applicable to that inversion. This holds since if $\mathcal{C}_2$ is not applicable to an inversion in λ, because of the shading, then the case for the meta pattern $\mathcal{C}_1$ applies to that inversion, and therefore it will remain an inversion after a pass through a stack. Applying $\mathcal{C}_2'$ to that inversion is therefore unnecessary.

We will denote the variation of $\mathsf{decorate}_{S_d}$, where non-exclusive shadings are applied, by $\mathsf{decorate}'_{S_d}$. Furthermore, we let preim'_{S_d} be the variation of preim_{S_d} that uses $\mathsf{decorate}'_{S_d}$.

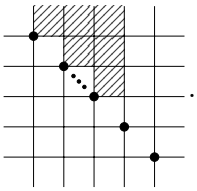
Corollary 2.1.3. *Stack-sortable permutations, with a stack of depth d, are precisely the elements of*

$$\mathsf{Av}(231, (d+1)d\ldots21).$$

Proof. We note that the stack-sortable permutations, with a stack of depth d, are the preimage of $\mathsf{Av}(21)$. We then have $\mathsf{cand}_d(21) = \{21\}$, and

$$\mathsf{decorate}'_{S_d}(d, 21, 21) = \{p_1, p_2\} = \left\{ \text{  }, \text{  } \right\}.$$

We have that $\mathsf{exp}(p_1) = 231$ and

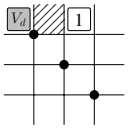
$\mathsf{exp}(p_2) =$  .

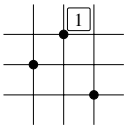
Now suppose we have an occurrence of $(d+1)d\dots1$, the underlying classical pattern of p_2, in a permutation π. If a shaded region in p_2 is occupied by an element of π, then π has an occurrence of 231. This holds since all shaded boxes in p_2 are located between descents in the underlying classical pattern. We therefore have that stack-sortable permutations, with a stack of depth d, are the elements of

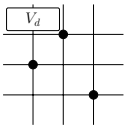
$$\mathsf{Av}(p_1, p_2) = \mathsf{Av}(231, (d+1)d\dots1). \qquad \square$$

Note that Goodrich et al. [9] independently discovered this corollary. This proposition generalizes Knuth's results, that stack-sortable permutations are the avoiders of 231 (we set $d = \infty$). Furthermore, we have reproduced the results of [8, Proposition 2], which states that bubble-sortable permutations are precisely the avoiders of 132 and 321.

To describe the permutations sortable by two passes through a stack of depth d, we need to describe the preimage of $\mathsf{Av}(231)$. In this case $\mathsf{cand}_d(231) = \{231, 321\}$ (if $d > 1$). Applying preim'_{S_d} then gives the following patterns.

$s_1 =$ 

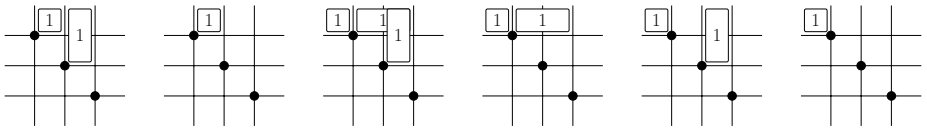
$s_2 =$ 

$s_3 =$ 

To complete the description, we also need to determine the preimage of $(d+1)d\dots1$. In general $\mathsf{preim}_{S_d}((d+1)d\dots1)$ yields a large set of patterns, which cannot easily be reduced. We will therefore only give the preimage when $d = 2$, i.e. the preimage of the bubble-sort operator.

If $d = 2$, we note that $V_2 =$  , which is contained in all non-empty permutations. We can therefore replace containment decorations, marked with V_2, with a marking and also replace any avoidance decoration, marked with V_2, with a shading. Applying preim'_{S_2} to 321 and replacing decorations marked with V_2, we then obtain the following pat-

terns.



Let A denote the set of these patterns. We note that $4321, 3421 \in A$ and each pattern in A either contains 4321 or 3421. Thus we have that $\mathsf{Av}(A) = \mathsf{Av}(4321, 321)$. Furthermore, we note that the shading and decoration in s_1 are redundant, since if a permutation π contains the underlying pattern of s_1, but an element of π lands in either the shaded region or the decorated region, then π contains either 4321 or 3421. Expanding the elements s_i then gives us that the permutations sortable by two passes through a stack of depth 2 are

$$S_2^{-2}(\mathsf{Av}(21)) = S_2^{-1}(\mathsf{Av}(231, 321)) = \mathsf{Av}(2341, 2431, 3241, 3421, 4231, 4321).$$

This is a special case of [8, Prop. 17] which gives a description of permutations sortable with k passes of bubble sort.

2.2 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 *enqueuing*, and the act of removing is called *dequeuing*.

Given an input permutation π, let $Q(\pi)$ be the permutation obtained by the following procedure.

1. If the queue is empty or its last element is smaller than the first element of the input, s, enqueue s.
2. Otherwise, dequeue elements from the queue and append them to the output permutation until the front element of the queue is larger than s. Append s to the output permutation.
3. Repeat this process until the input is empty.
4. Empty the queue into the output permutation.

We call Q the *queue-sort operator*, and for $\pi \in \mathfrak{S}_n$, if $Q(\pi) = \mathsf{id}_n$, we say that π is *queue-sortable*. Strictly speaking, we have described the process of sorting with two parallel queues (or a queue with a bypass). However, for simplicity, we refer to this process as queue sort.

Figure 2.3 shows the result of applying the queue-sort operator to the permutation 45321. First 2, 3 and 5 are enqueued. Since 4 is smaller than the back of the queue, all elements larger than 4 are dequeued into the output, and 4 is subsequently appended to the output. The 1 is smaller than the 5 on the queue, and also bypasses the queue. Finally, the 6 is added to the queue and the queue is then emptied into the output.

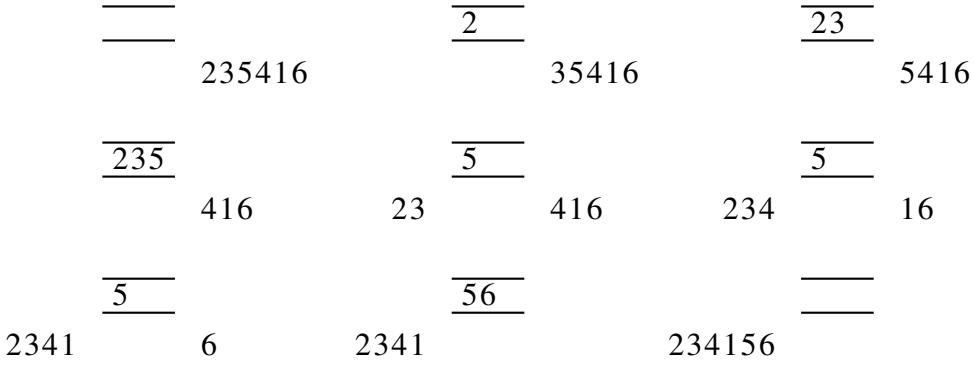


Figure 2.3: Queue-sort operator applied to 235416

We will now give an algorithm preim_Q for describing preimages of pattern classes, defined by the avoidance of a classical pattern, under the queue-sort operator. This algorithm proceeds in two steps, similar to preim_{S_d}. Given a target pattern p, we first generate candidate patterns that can possibly become p after a single pass through a queue and then add decorations to the candidates, if possible.

We note that a non-inversion in a permutation must remain a non-inversion after a pass through a queue. For a permutation π, we therefore have that $\mathsf{inv}(Q(\pi)) \subseteq \mathsf{inv}(\pi)$. Thus we can consider the candidates

$$\mathsf{gcand}(p) = \{\lambda : \mathsf{inv}(p) \subseteq \mathsf{inv}(\lambda)\}.$$

Similar to preim_{S_d}, in the second step of preim_Q, we must consider all inversions in each candidate. Suppose we have a candidate $\lambda \in \mathsf{gcand}(p)$ and a permutation π containing λ. An inversion in λ, corresponding to the letters a and b in π, will remain an inversion after a pass through the queue if either of the two cases below applies.

1. There is an element c, larger than a, that appears before a and b in π. In this case c is enqueued first. When a becomes the first element of the input, either c or an element

larger than c is at the end of the queue, and therefore a is appended to the output. When b becomes the first element of the output, either c, or an element larger than c, is at the end of the queue, and therefore b is appended to the output. The relative order of a and b therefore remains the same after applying the queue-sort operator. This case corresponds to the meta pattern $\mathcal{Q}_1$ in Table 2.2.

2. There are elements $c > d$, that appear between a and b in π, in that order. If a is not enqueued, then a will appear before b in the output. Suppose, therefore, that a is enqueued. When d becomes the first element of the input, it is appended to the output, since c, or an element larger than c, is the last element in the queue. Before d is appended to the output, a must be appended to the output, since if a is in the queue, it will be dequeued before d is appended to the output. The relative order of a and b will therefore remain the same after a pass through a queue. This case corresponds to the meta pattern $\mathcal{Q}_2$ in Table 2.2. Note that the shading is not necessary, but is added to exclude the first case.

If neither of the two cases applies to an inversion in λ, it becomes a non-inversion in the output. This corresponds to the meta pattern $\mathcal{Q}_3$ in Table 2.2. Algorithm 2.2 gives a formal description of $\mathsf{decorate}_Q$.

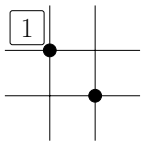
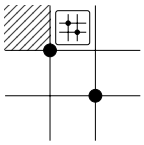
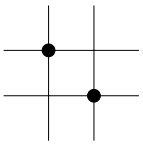
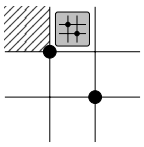
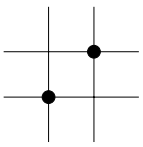
Candidates		**Target**
$\mathcal{Q}_1 =$ 	$\mathcal{Q}_2 =$ 	
$\mathcal{Q}_3 =$ 		

Table 2.2: Meta patterns applied to inversions in candidates to obtain inversion or non-inversions in the target after a pass through a queue

We then define

$$\mathsf{preim}_Q(p) = \bigcup_{\lambda \in \mathsf{gcand}(p)} \mathsf{decorate}_Q(p, \lambda),$$

and also define $\mathsf{decorate}'_Q$ as the variation of $\mathsf{decorate}_Q$ that does not apply exclusive shadings, similar to $\mathsf{decorate}'_{S_d}$, and we define preim'_Q analogously to preim'_{S_d}.

Algorithm 2.2: $\mathsf{decorate}_Q$

Input: A target pattern p; and $\lambda \in \mathsf{gcand}(p)$
Output: A (possibly empty) set of decorated patterns M

```
Let M = {(λ, ∅, ∅, ∅, ∅)};
for v ∈ inv(λ) do
    for r ∈ M do
        Remove r from M;
        if v ∈ inv(p) then
            for i ∈ {1, 2} do
                if Q_i is applicable to v in r then apply Q_i to v in r and add to M;
        else
            if Q_3 is applicable to v in r then apply Q_3 to v in r and add to M;
return M;
```

Theorem 2.2.1. *The preimage of the class of permutations, defined by the avoidance of p, under the queue-sort operator is*

$$Q^{-1}(\mathsf{Av}(p)) = \mathsf{Av}(\mathsf{preim}_Q(p)).$$

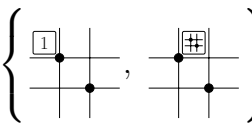
Proof. The proof is analogous to the proof of Theorem 2.1.2. □

Corollary 2.2.2. *The permutations sortable by k passes through a queue are precisely the elements of*

$$\mathsf{Av}((k+2)(k+1)\dots 1).$$

Proof. We will prove our proposition by induction on k. For $k = 1$, we are interested in the preimage of $\mathsf{Av}(21)$, i.e., the permutations sortable with a single pass through a queue. We then have $\mathsf{gcand}(21) = \{21\}$, and therefore

$$\mathsf{preim}'_Q(21) = \left\{ \right\} = \{321, 2431\}.$$

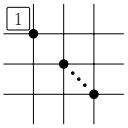


Now, since 2431 contains the pattern 321, we have that $\mathsf{Av}(321, 2431) = \mathsf{Av}(321)$. Thus the permutations sortable by a single pass through a queue are $Q^{-1}(\mathsf{Av}(21)) = \mathsf{Av}(321)$.

We now want to describe the permutations sortable by k passes through a queue. By the induction hypothesis, we know that the permutations sortable by $k-1$ passes through a queue are $\mathsf{Av}(r)$, where $r = (k+1)k\dots 1$. The permutations sortable by k passes through a queue are therefore $Q^{-1}(\mathsf{Av}(r))$. First we note that $\mathsf{gcand}(r) = \{r\}$, so we need only apply $\mathsf{decorate}_Q$ to r. Furthermore, since all inversions in the candidate must become

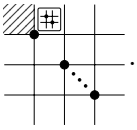
inversions in the target, we need only consider the cases for $\mathcal{Q}_1$ and $\mathcal{Q}_2$. If we consider the first inversion in r, i.e., $((k+1), k)$, then both cases are applicable. Let us consider these two cases.

1. If we apply $\mathcal{Q}_1$ to the first inversion, we obtain

$$r_1 = (r, \varnothing, \{(\{(0, k+1)\}, 1)\}) =$$  .

   We now note that after this application, we cannot apply $\mathcal{Q}_2$ to any inversion in r_1, since the shading of $\mathcal{Q}_2$ will cover the marking of r_1. Furthermore, we note that applying $\mathcal{Q}_1$ to any other inversion in r_1 has no effect since the marking in r_1 is contained in the marking added by applying $\mathcal{Q}_1$ to any other inversion in r_1, and the new markings are therefore redundant.

2. If we apply $\mathcal{Q}_2$ to the first inversion, we obtain

$$r_2 = (r, \{(0, k+1)\}, \varnothing, \varnothing, \{(\{(1, k+1)\}, 21)\}) =$$  .

   After this application, we note that we cannot apply $\mathcal{Q}_1$ to an inversion $(k+1, i)$ in r_2, since that would imply marking the box $(0, k+1)$ which is shaded in r_2. Furthermore, we note that applying $\mathcal{Q}_2$ to an inversion $(k+1, i)$ in r_2 has no effect since the containment decoration in r_2 is contained in the containment decoration added by such an application, and therefore the new decoration is unnecessary. For all other inversions in r_2, i.e., inversions (i, j) where $i < k+1$, we have that $\mathcal{Q}_2$ is not applicable, since applying $\mathcal{Q}_2$ to (i, j) would imply shading over the element $k+1$ in r_2. Furthermore, we note that applying $\mathcal{Q}_1$ to (i, j) has no effect, since the element $k+1$ in r_2 is contained in the marking added by the application, and the marking is therefore redundant.

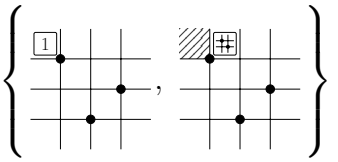
We therefore have that $\mathsf{preim}_Q((k+1)k\ldots1) = \{r_1, r_2\}$. We note that since r_2 contains r_1, containment of r_2 implies containment of r_1. Therefore, we have that $\mathsf{Av}(r_1, r_2) = \mathsf{Av}(r_1)$ and thus the permutations sortable by k passes through a queue are

$$Q^{-1}(\mathsf{Av}((k+1)k\ldots1)) = \mathsf{Av}(r_1) = \mathsf{Av}((k+2)(k+1)\ldots1). \qquad \square$$

Example 2.2.3. Suppose we want to find the preimage of $\mathsf{Av}(312)$ under the queue-sort operator, i.e., $Q^{-1}(\mathsf{Av}(312))$. We have that $\mathsf{gcand}(312) = \{321, 312\}$. Applying

$\mathsf{decorate}_Q$ to 321 yields no patterns, since the inversion $(2,1)$ in 321 must become a non-inversion after a pass through a queue. However, the meta pattern $\mathcal{Q}_3$ is not applicable to $(2,1)$ in 321 since in order to apply it we must shade over the element 3. We therefore have

$$\mathsf{preim}_Q(312) = \mathsf{decorate}_Q(312, 312) = \{p_1, p_2\} =$$



However, we note that $p_1 = 4312$ is contained in p_2, and therefore we have that

$$Q^{-1}(\mathsf{Av}(312)) = \mathsf{Av}(4312).$$

2.3 Sorting with a pop-stack

A *pop-stack*, introduced by Avis and Newborn in [10], is a stack of unlimited depth, with the added restriction that when the stack is popped, all the elements must be removed from the stack.

Given an input permutation π, let $T(\pi)$ be the permutation obtained by the following procedure:

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 all the elements from the stack and append them to the output permutation. Push s onto the stack.
3. Repeat this process until the input is empty.
4. Pop all elements off the stack and append them to the output permutation.

We call T the *pop-stack-sort operator*, and for $\pi \in \mathfrak{S}_n$, if $T(\pi) = \mathsf{id}_n$, we say that π is *pop-stack-sortable*.

Figure 2.4 shows the result of applying the pop-stack-sort operator to the permutation 51243. First 5 and 1 are pushed onto the stack. Since 2 is larger than 1, the stack must be emptied and 2 is subsequently pushed onto the stack. The 2 is then popped off the stack, and both 4 and 3 are pushed onto the stack. Finally the stack is emptied into the output.

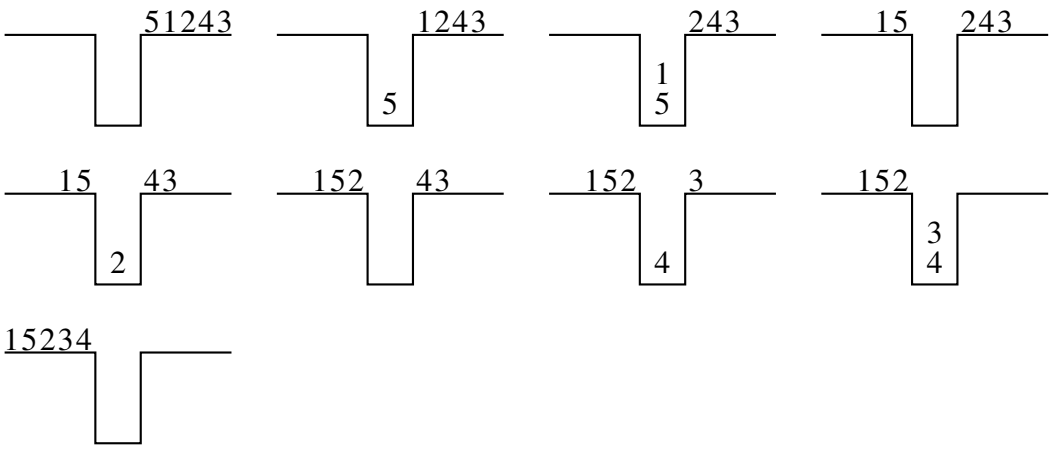


Figure 2.4: Pop-stack-sort operator applied to 51243

The preimage algorithm for T, preim_T, proceeds in two steps similar to preim_{S_d}. Since non-inversions remain non-inversions after a pass through a pop-stack, we consider the candidates of gcand. For each candidate we then consider all inversions. Suppose we have a candidate $\lambda \in \mathsf{gcand}(p)$ and a permutation π containing λ. An inversion in λ, corresponding to the letters a and b in π, will remain an inversion after a pass through the pop-stack if one of the three cases below applies.

1. There is an element c, larger than a, that appears between a and b in π. In this case a is pushed onto the stack, and subsequently popped off by c, before b is pushed onto the stack. The relative order of a and b therefore remains the same after applying the pop-stack-sort operator. This corresponds to the meta pattern $\mathcal{T}_1$ in Table 2.3.

2. There is an element c, smaller than b, that appears between a and b in π. In this case c is pushed onto the stack, and subsequently popped off by b. Since all elements must be popped off the stack when c is popped, a will be popped off the stack before b. The relative order of a and b therefore remains the same. This corresponds to the meta pattern $\mathcal{T}_2$ in Table 2.3. Note that the shading is not necessary, but is added to exclude the first case.

3. There is a non-inversion that appears between a and b in π, in which both elements are larger than b and smaller than a. In this case the non-inversion will cause the stack to be popped, and therefore a is popped off the stack before b. The relative order of a and b therefore remains the same. This corresponds to the meta pattern $\mathcal{T}_3$ in Table 2.3. Note that the shadings are not necessary, but are added to exclude the first two cases.

If none of the three cases applies to an inversion in λ, it becomes a non-inversion in the output. This corresponds to the meta pattern $\mathcal{T}_4$ in Table 2.2. Algorithm 2.3 gives a formal description of $\mathsf{decorate}_T$.

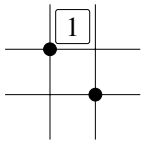
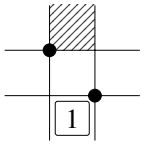
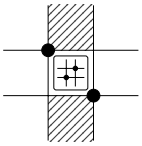
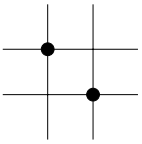
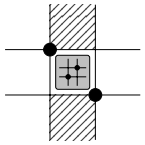
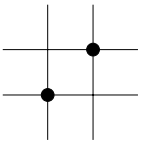
Candidates	**Target**
$\mathcal{T}_1 =$  , $\mathcal{T}_2 =$  , $\mathcal{T}_3 =$ 	
$\mathcal{T}_4 =$ 	

Table 2.3: Meta patterns applied to inversions in candidates to obtain inversions or non-inversions in the target after a single pass through a pop-stack

Algorithm 2.3: $\mathsf{decorate}_T$

Input: A target pattern p; and $\lambda \in \mathsf{gcand}(p)$
Output: A (possibly empty) set of decorated patterns M

```
Let M = {(λ, ∅, ∅, ∅, ∅)};
for v ∈ inv(λ) do
    for r ∈ M do
        Remove r from M;
        if v ∈ inv(p) then
            for i ∈ {1, 2, 3} do
                if T_i is applicable to v in r then apply T_i to v in r and add to M;
        else
            if T_4 is applicable to v in r then apply T_4 to v in r and add to M;
return M;
```

We then define

$$\mathsf{preim}_T(p) = \bigcup_{\lambda \in \mathsf{gcand}(p)} \mathsf{decorate}_T(p, \lambda),$$

and, as before, we let $\mathsf{decorate}'_T$ and preim'_T be the variations of $\mathsf{decorate}_T$ and preim_T that do not use exclusive shadings.

Theorem 2.3.1. *The preimage of the class of permutations, defined by the avoidance of p, under the pop-stack-sort operator is*

$$T^{-1}(\mathsf{Av}(p)) = \mathsf{Av}(\mathsf{preim}_T(p)).$$

Proof. The proof is analogous to the proof of Theorem 2.1.2. □

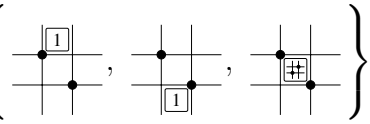
Using preim_T, we can now describe the pop-stack-sortable permutations, which was first done by Avis and Newborn [10].

Corollary 2.3.2. *The pop-stack-sortable permutations are precisely the elements of*

$$\mathsf{Av}(231, 312).$$

Proof. We have that $\mathsf{gcand}(21) = \{21\}$ and therefore

$$\begin{aligned}\mathsf{preim}_T(21) &= \{p_1, p_2, p_3\} \\ &= \left\{ \right. \\ &= \{231, 312, 4231\}.\end{aligned}$$



Since 231 is contained in 4231, we have that the permutations sortable by a single pass through a pop-stack are

$$T^{-1}(\mathsf{Av}(21)) = \mathsf{Av}(231, 312, 4231) = \mathsf{Av}(231, 312).$$ □

2.4 Insertion sort

Let $\pi = \pi_1\pi_2 \dots \pi_n$ be a permutation, i be the leftmost descent in π and $j \leq i$ be the leftmost index such that $\pi_j > \pi_{i+1}$. We then define the *insertion-sort operator* as

$$I(\pi) = \pi_1\pi_2 \dots \pi_j\pi_{i+1}\pi_{j+1} \dots \pi_i\pi_{i+2} \dots \pi_n.$$

In the case when π has no descents, i.e., when $\pi = \mathsf{id}_n$, we define $I(\mathsf{id}_n) = \mathsf{id}_n$. If $I(\pi) = \mathsf{id}_n$, we say that π is *insertion-sortable*.

The definition of I is equivalent to the process of finding the first descent in π and inserting the descent bottom in the proper place to the left of its position. For example, if we apply the insertion-sort operator to $\pi = 13524$, we obtain $I(\pi) = 12354$. The first descent in π is 3, i.e., in position 3 we have a 5 followed by a 2. The 2 is then moved to the left of its position and inserted into π such that the first 4 elements of π are in ascending order, i.e. between the 1 and the 3.

We note that after applying the insertion-sort operator to a permutation, its non-inversions remain the same. When finding candidates for the preimage of I we therefore use gcand. For each candidate we then consider all inversions. Suppose we have a candidate $\lambda \in \mathsf{gcand}(p)$ and a permutation π containing λ. An inversion in λ, corresponding to the letters a and b in π, will remain an inversion after a pass through insertion sort if one of the three cases below applies.

1. There is an element c, larger than a, that appears before a and b in π. In this case there is a descent to the left of b in π. Since b is not the descent bottom of this descent, b will not be moved when insertion sort is applied, and the relative order of a and b will therefore remain the same. This corresponds to the meta pattern $\mathcal{I}_1$ in Table 2.4.

2. There is an inversion that appears before both a and b. In this case there is a descent in π to the left of a, whose descent bottom will be moved left, leaving a and b untouched. This corresponds to the meta pattern $\mathcal{I}_2$ in Table 2.4. Note that the shading is not necessary, but is added to exclude the first case.

3. There is an element $c < a$ that appears between a and b in π. In this case there will also be a descent to the left of b. This corresponds to the meta pattern $\mathcal{I}_3$ in Table 2.4. Note that the avoidance decoration, forbidding an occurrence of 21, and the shading are not necessary, but are added to exclude the first two cases.

4. There is an inversion (c, d) in π that appears between a and b, with $d > a$. In this case there is a descent in π to the left of b, whose descent bottom will be moved left, leaving b untouched. This corresponds to the meta pattern $\mathcal{I}_4$ in Table 2.4. Note that the avoidance decoration, forbidding an occurrence of 21, and the shadings are not necessary, but are added to exclude the first three cases.

If none of these four cases applies to an inversion in λ, it becomes a non-inversion in the output. This corresponds to the meta pattern $\mathcal{I}_5$ in Table 2.2. Algorithm 2.4 gives a formal description of $\mathsf{decorate}_I$.

We then define

$$\mathsf{preim}_I(p) = \bigcup_{\lambda \in \mathsf{gcand}(p)} \mathsf{decorate}_I(p, \lambda),$$

and also define $\mathsf{decorate}'_I$ and preim'_I as the variation of $\mathsf{decorate}_I$ and preim_I that do not use exclusive shadings or avoidance decorations.

Algorithm 2.4: $\mathsf{decorate}_I$

Input: A target pattern p; and $\lambda \in \mathsf{gcand}(p)$
Output: A (possibly empty) set of decorated patterns M

```
Let M = {(λ, ∅, ∅, ∅, ∅)};
for v ∈ inv(λ) do
    for r ∈ M do
        Remove r from M;
        if v ∈ inv(p) then
            for i ∈ {1, 2, 3, 4} do
                if I_i is applicable to v in r then apply I_i to v in r and add to M;
        else
            if I_5 is applicable to v in r then apply I_5 to v in r and add to M;
return M;
```

Theorem 2.4.1. *The preimage of the class of permutations, defined by the avoidance of p, under the insertion-sort operator is*

$$I^{-1}(\mathsf{Av}(p)) = \mathsf{Av}(\mathsf{preim}_I(p)).$$

Proof. The proof is analogous to the proof of Theorem 2.1.2. □

Proposition 2.4.2. *The insertion-sortable permutations are precisely the elements of*

$$\mathsf{Av}(312, 321, 2143).$$

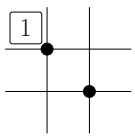
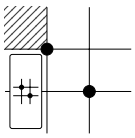
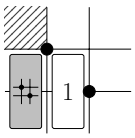
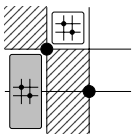
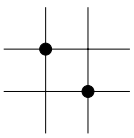
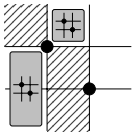
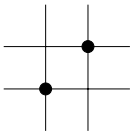
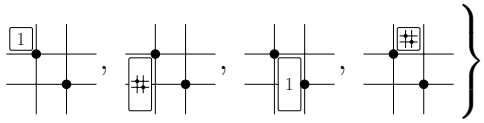
Candidates	**Target**
$\mathcal{I}_1 =$  $, \mathcal{I}_2 =$  $, \mathcal{I}_3 =$  $, \mathcal{I}_4 =$ 	
$\mathcal{I}_5 =$ 	

Table 2.4: Meta patterns applied to inversions in candidates to obtain inversions or non-inversions in the target after applying insertion sort

Proof. We have that $\mathsf{gcand}(21) = \{21\}$ and thus we have

$$\mathsf{preim}'_I(21) = \left\{ \right.$$

$$\left. \right\}$$

$$= \{321, 3241, 3142, 2143, 312, 2431\}.$$

However, we note that 321 is contained in 3241 and 2431 and that 312 is contained 3142. Therefore, we have that the permutations sortable by a single pass through insertion sort are

$$I^{-1}(\mathsf{Av}(21)) = \mathsf{Av}(312, 321, 2143). \qquad \square$$

2.5 Pancake sort

A *prefix reversal* of a permutation $\pi = \pi_1\pi_2 \dots \pi_n$ is the process of selecting an index i and reversing the first i elements of π, resulting in the permutation $\pi_i\pi_{i-1} \dots \pi_1\pi_{i+1} \dots \pi_n$. *Pancake sort* is a sorting algorithm where the only allowed operation is prefix reversal. The name of the algorithm is derived from the way the algorithm is often visualized. We imagine we have a stack of pancakes, where no two have the same diameter. Our aim is to sort the pancakes in such a way that the diameter of pancakes increases from top to bottom. The only allowed operation is to insert a spatula somewhere in the stack and use it to *flip* all the pancakes on top of the spatula.

A few pancake-sorting algorithms have been proposed that aim to minimize the number of prefix reversals (see e.g. [11] and [12]). Finding the minimum number of flips needed to pancake-sort a permutation has, however, been proven to be NP-hard [13]. We will use a simple deterministic pancake-sorting algorithm defined as follows. Let $\pi \in \mathfrak{S}_n$ such that $\pi = \alpha k \beta (k+1)(k+2) \dots n$ and the last element of β is not a fixed point of π (β is therefore non-empty). We then define the *pancake-sort operator* as

$$P(\pi) = \beta^r \alpha k (k+1)(k+2) \dots n.$$

Furthermore, if no such β exists, then $\pi = \mathsf{id}_n$ and we define $P(\mathsf{id}_n) = \mathsf{id}_n$. If $P(\pi) = \mathsf{id}_n$, we say that π is *pancake-sortable*.

This definition is equivalent to the following procedure (described in terms of pancakes).

1. Find the largest pancake that is out of place.

2. Flip the stack at that pancake's position, so that it moves to the top of the stack.
3. Flip the stack at the position where the pancake that is now on top of the stack is supposed to be.

Figure 2.5 shows the result of applying the stack-sort operator to the permutation 214356. First we locate the largest element that is out of place, in this case 4, located at position 3. We then flip the stack at position 3, so the element 4 is now on top of the stack. Finally, we flip the stack at position 4. Since the element 4 was on top of the stack, after the flip, it ends in its correct position.

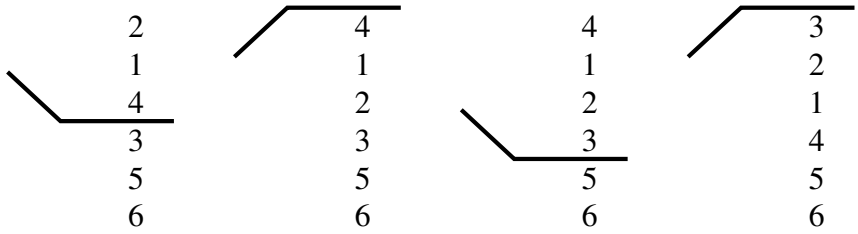


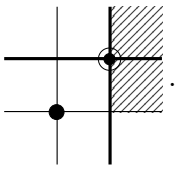
Figure 2.5: Pancake-sort operator applied to 214356

We note that the pancake-sort operator does not preserve non-inversions, unlike the sorting operators we have explored so far. For example, in the definition above, if there is a non-inversion in β, then it will become an inversion after the operator is applied. We therefore consider all elements of $\mathfrak{S}_n$ as candidates for a preimage for any pattern. For the same reason, when applying restrictions to the candidates, we must now not only consider inversions in the candidates, but also non-inversions.

Here, however, we face a problem. In order to determine whether or not an inversion becomes a non-inversion, and vice versa, we must be able to determine where the inversion is placed relative to α and β. To determine this placement, we must know where k is placed in the pattern. There is, however, no guarantee that k will correspond to any element of a contained pattern in π. Thus, if we have an inversion in a candidate pattern, we have no way of determining where the elements in π, corresponding to the inversion in the pattern, will be placed relative to α and β. To overcome this problem, we introduce a more expressive notion of a meta pattern.

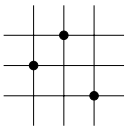
2.5.1 More expressive meta patterns

Our more expressive notion of a meta pattern is best demonstrated by an example:

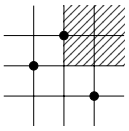
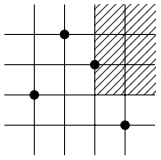
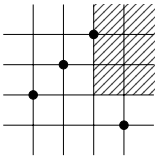
$\mathcal{P} =$  .

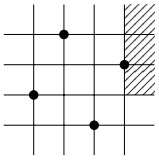
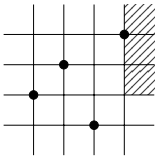
Here we introduce a new attribute to meta patterns: circled dots (which appear on thick lines). As in our previous definition of a meta pattern, the thin lines and solid dots correspond to elements in the pattern to which the meta pattern is applied. When applied to a pattern, a circled point can either match a point in the pattern, or a point can be added to the pattern to match the circled point, as long as all constraints are satisfied. When applying a meta pattern with circled dots, we can therefore obtain more than one pattern.

If we apply $\mathcal{P}$ to the element 2 in

$p =$ 

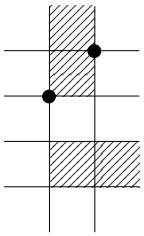
we obtain the patterns

$p_1 =$  , $p_2 =$  , $p_3 =$  ,

$p_4 =$  , $p_5 =$  .

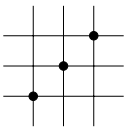
The pattern p_1 is the result of using the element 3 in p as the circled point. The patterns p_2 and p_3 are the results of inserting the circled point into boxes $(2, 2)$ and $(2, 3)$ in p, respectively, and p_4 and p_5 are obtained by inserting into boxes $(3, 2)$ and $(3, 3)$. Note that the circled point cannot be in boxes $(1, 2)$ or $(1, 3)$ since the added shading would then cover the element 3 of p.

As another example, let

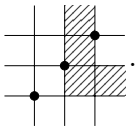
$\mathcal{Q} =$  .

When we apply this meta pattern, we need to specify which lines we apply the pattern to, as well as which element, since the pattern contains thin lines with no points.

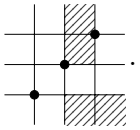
If we apply $\mathcal{Q}$ to 23 in

$q =$ 

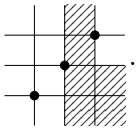
where $\mathcal{Q}_1^h$ and $\mathcal{Q}_2^h$ correspond to q_1^h and q_2^h, we obtain

 .

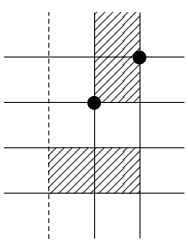
We can also let $\mathcal{Q}_1^h$ and $\mathcal{Q}_2^h$ correspond to q_0^h and q_1^h, and obtain

 .

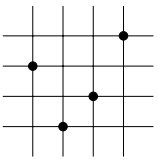
Furthermore, we can let $\mathcal{Q}_1^h$ and $\mathcal{Q}_2^h$ correspond to q_0^h and q_2^h, and obtain

 .

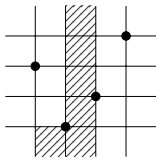
We can also require lines in meta patterns to have certain attributes. We draw those lines as dotted. For example, let

$\mathcal{R} =$  ,

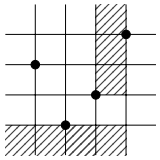
where the dotted line is the line containing the rightmost point larger than $\mathcal{R}_4^h$, or the beginning of the pattern if no such point exists. Applying $\mathcal{R}$ to 12 in

$r =$  ,

where $\mathcal{R}_1^h$ and $\mathcal{R}_2^h$ correspond to r_0^h and r_1^h, then yields

 ,

since the dotted line corresponds to r_1^v, as it contains the rightmost element, to the left of 1, larger than 2. However, if we apply $\mathcal{R}$ to 24 in r and let $\mathcal{R}_1^h$ and $\mathcal{R}_2^h$ correspond to r_0^h and r_1^h, we obtain

 ,

since the dotted line corresponds to the r_0^v, as there is no element larger than 4 to the left of 2. Note that meta patterns containing dotted lines will not be used in this section.

With these more expressive meta patterns, we are now able to describe the preimage of the pancake-sort operator. Suppose we have a candidate pattern $\lambda = \lambda_1\lambda_2\ldots\lambda_m$ for a target pattern p. Furthermore, suppose we have a permutation π that contains λ, where we can write

$$\pi = \alpha k \beta \gamma,$$

where $\gamma = (k+1)(k+2)\ldots n$, and the last element of β is not a fixed point of π. By the definition of the pancake-sort operator, we then have

$$P(\pi) = \beta^r \alpha k (k+1)\ldots n.$$

Suppose, finally, that we have two indices $i < j$, and let a and b be the elements in π corresponding to λ_i and λ_j, respectively. We then consider the following cases.

1. An inversion in λ remains an inversion in p, i.e., $(\lambda_i, \lambda_j) \in \mathsf{inv}(\lambda)$ and $(\lambda_i, \lambda_j) \in \mathsf{inv}(p)$. This can only happen if both a and b are in α, since α remains the same after P has been applied to π. If γ is empty, then $k = n$ corresponds to the largest

element in π. We represent this by the meta pattern $\mathcal{P}_1$ in Table 2.5. We add a circled point to represent k, since it is not guaranteed that k will correspond to an element of λ. We then add shadings above k, since k is the largest element outside γ. The case when γ is non-empty is covered by the meta pattern $\mathcal{P}_2$. We consider the leftmost point in γ, which is larger than k, and add that as a circled point to the pattern since it is not guaranteed that it will correspond to an element in the pattern. Since γ must be in increasing order, we add a restrictive decoration, forbidding 21 above and to the right of the leftmost point of γ, which corresponds to the rest of γ. We then add shadings above k, since k is the largest element outside γ. We also add a shading below the leftmost element of γ, since γ does not contain an element smaller than k. Finally we add a marking to the right of the element corresponding to k, since β cannot be non-empty.

We must consider the two cases $\gamma = \varepsilon$ and $\gamma \neq \varepsilon$ separately, since we want to represent the "boundary" of γ in the meta patterns. The boundary of γ is the end of the pattern, in the case when γ is empty. In the case when γ is non-empty, the boundary is defined by the leftmost point in γ, which is represented by the right circled point in $\mathcal{P}_2$.

2. A non-inversion in λ becomes an inversion in p, i.e., we have $(\lambda_i, \lambda_j) \in \mathsf{ninv}(\lambda)$ and $(\lambda_i, \lambda_j) \in \mathsf{inv}(p)$. This can happen for two reasons.

   (a) The subword β contains both a and b. Since β is reversed after P has been applied to π, the relative order of a and b will change. This case is represented by the meta patterns $\mathcal{P}_9$ and $\mathcal{P}_{10}$ in Table 2.6. Again, the left circled point corresponds to k, so the region to the left of the circled point corresponds to α and the region to the right of it, up to the shaded region, corresponds to β. The meta pattern $\mathcal{P}_9$ covers the case when γ is empty and $\mathcal{P}_{10}$ when γ is non-empty.

   (b) The letter a is in α and b is in β. Since the relative order of α and β is reversed, after P has been applied to π, the relative order of a and b will change. This case is represented by the meta patterns $\mathcal{P}_{11}$ and $\mathcal{P}_{12}$.

3. An inversion in λ becomes a non-inversion in p, i.e., $(\lambda_i, \lambda_j) \in \mathsf{inv}(\lambda)$ and $(\lambda_i, \lambda_j) \in \mathsf{ninv}(p)$. There are three cases where this can happen.

   (a) As shown in the previous case, if subword β contains both a and b or a is in α and b is in β, the relative order of a and b will change. The first case is represented by the meta patterns $\mathcal{P}_3$ and $\mathcal{P}_4$ and the second case by $\mathcal{P}_5$ and $\mathcal{P}_6$.

(b) We have $a = k$ and b is in β. Since β is moved to the left of k when P is applied to π, the relative order of a and b changes. This corresponds to the meta patterns $\mathcal{P}_7$ and $\mathcal{P}_8$. The meta pattern $\mathcal{P}_7$ covers the case when γ is empty and $\mathcal{P}_8$ the case when γ is non-empty. In both meta patterns the point representing the element corresponding to k is not circled, since in this case λ_i corresponds to k in π.

4. A non-inversion in λ remains a non-inversion in p, i.e. $(\lambda_i, \lambda_j) \in \mathsf{ninv}(\lambda)$ and $(\lambda_i, \lambda_j) \in \mathsf{ninv}(p)$. There are three cases where this can happen.

   (a) The subword α contains both a and b. Since α remains unchanged after P has been applied to π, the relative order of a and b remains the same. This corresponds to the meta patterns $\mathcal{P}_{13}$ and $\mathcal{P}_{14}$.

   (b) The subword α contains a and $b = k$. Since the relative order of α and k remains the same, after P has been applied to π, the relative order of a and b also remains the same. This corresponds to the meta patterns $\mathcal{P}_{15}$ and $\mathcal{P}_{16}$.

   (c) The letter b is in γ. No matter where a is to the left of b, since γ remains unchanged, after P has been applied to π, the relative order of a and b will remain the same. This corresponds to the meta pattern $\mathcal{P}_{17}$. The avoidance decoration above and to the right of the right point, along with the shading below and to the left of it, guarantee that it is in γ.

For a permutation $p \in \mathfrak{S}_n$, we define

$$\mathsf{preim}_P(p) = \bigcup_{\lambda \in \mathfrak{S}_n} \mathsf{decorate}_P(p, \lambda),$$

and present the following theorem.

Theorem 2.5.1. *The preimage of the class of permutations, defined by the avoidance of p, under the pancake-sort operator is*

$$P^{-1}(\mathsf{Av}(p)) = \mathsf{Av}(\mathsf{preim}_P(p)).$$

Proof. The proof is analogous to the proof of Theorem 2.1.2. □

Corollary 2.5.2. *The pancake-sortable permutations are precisely the elements of*

$$\mathsf{Av}(132, 312, 3241).$$

Proof. We begin by applying the preim_P to 21 to obtain the preimage of $\mathsf{Av}(21)$. The candidates we use are $\mathfrak{S}_2 = \{12, 21\}$. In $\mathsf{decorate}_P(21, 12)$ we apply the meta patterns

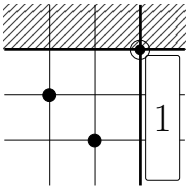
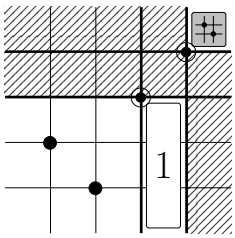
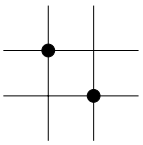
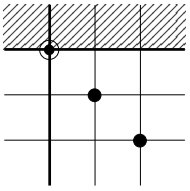
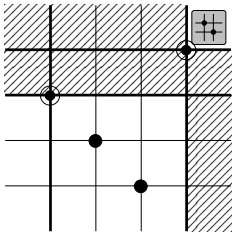
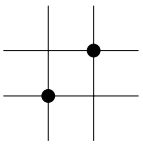
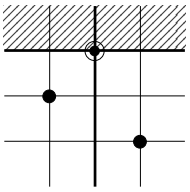
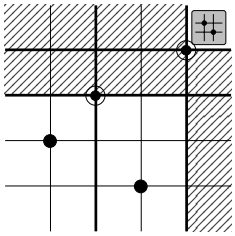
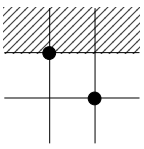
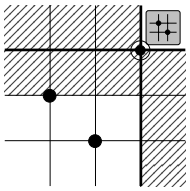
Candidates	**Target**
$\mathcal{P}_1 =$  , $\mathcal{P}_2 =$ 	
$\mathcal{P}_3 =$  , $\mathcal{P}_4 =$  ,	
$\mathcal{P}_5 =$  , $\mathcal{P}_6 =$  ,	
$\mathcal{P}_7 =$  , $\mathcal{P}_8 =$ 	

Table 2.5: Meta patterns applied to inversions in candidates to obtain inversions or non-inversions in the target after pancake-sorting

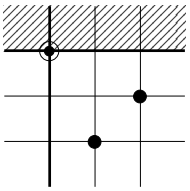
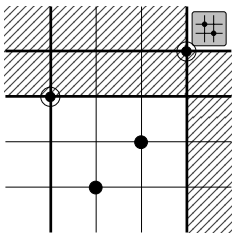
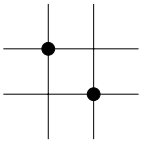
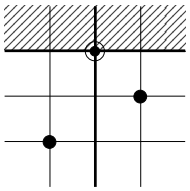
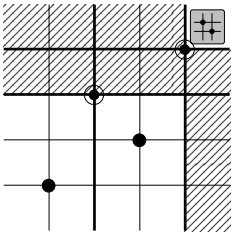
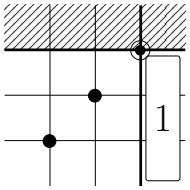
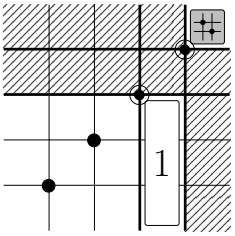
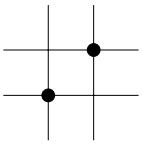
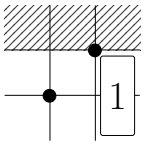
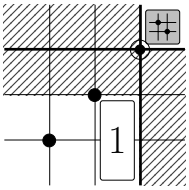
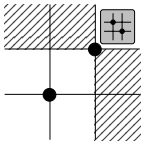
Candidates	**Target**
$\mathcal{P}_9 =$  , $\mathcal{P}_{10} =$  ,	
$\mathcal{P}_{11} =$  , $\mathcal{P}_{12} =$ 	
$\mathcal{P}_{13} =$  , $\mathcal{P}_{14} =$  ,	
$\mathcal{P}_{15} =$  , $\mathcal{P}_{16} =$  ,	
$\mathcal{P}_{17} =$ 	

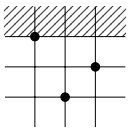
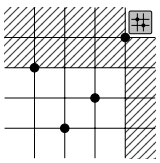
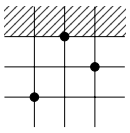
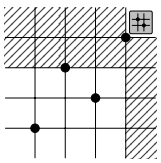
Table 2.6: Meta patterns applied to non-inversions in candidates to obtain inversions or non-inversions in the target after pancake-sorting

Algorithm 2.5: $\mathsf{decorate}_P$

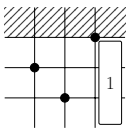
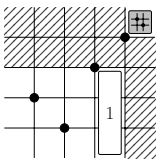
Input: A target pattern $p \in \mathfrak{S}_n$; and $\lambda \in \mathfrak{S}_n$
Output: A (possibly empty) set of decorated patterns M

```
Let M = {(λ, ∅, ∅, ∅, ∅)};
for v ∈ inv(λ) do
    for r ∈ M do
        Remove r from M;
        if v ∈ inv(p) then
            for i ∈ {1, 2} do
                if P_i is applicable to v in r then apply P_i to v in r and add to M;
        else
            for i ∈ {3, 4, 5, 6, 7, 8} do
                if P_i is applicable to v in r then apply P_i to v in r and add to M;
for v ∈ ninv(λ) do
    for r ∈ M do
        Remove r from M;
        if v ∈ inv(p) then
            for i ∈ {9, 10, 11, 12} do
                if P_i is applicable to v in r then apply P_i to v in r and add to M;
        else
            for i ∈ {13, 14, 15, 16, 17} do
                if P_i is applicable to v in r then apply P_i to v in r and add to M;
return M;
```

$\mathcal{P}_9, \ldots, \mathcal{P}_{12}$ and obtain

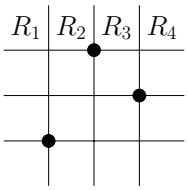
$p_1 =$  , $p_2 =$  , $p_3 =$  , $p_4 =$  .

In $\mathsf{decorate}_P(21, 21)$ we apply the meta patterns $\mathcal{P}_1$ and $\mathcal{P}_2$ and obtain

$p_4 =$  , $p_5 =$  .

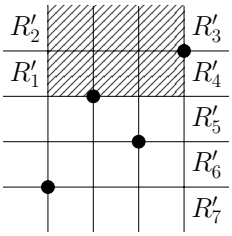
Let A denote the set of all p_i and let $B = \{132, 312, 3241)$. We now want to show that $\mathsf{Av}(A) = \mathsf{Av}(B)$, or equivalently that $\mathsf{Co}(A) = \mathsf{Co}(B)$. Since 312 is contained in p_1 and p_2, 132 is contained in p_3 and p_4 and either 132 or 3241 is contained p_4 and p_5, we have

that $\mathsf{Co}(A) \subseteq \mathsf{Co}(B)$. To complete the poof we therefore need to show that if we have a permutation $\pi \in \mathsf{Co}(B)$ then $\pi \in \mathsf{Co}(A)$. Now, suppose we have a permutation π that contains 132 and let the following graph denote an occurrence of 132 in π such that there is no other occurrence of 132 in π with a larger element corresponding to 3 in the pattern.

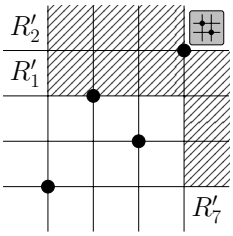


For the sake of brevity, we will talk about points in the regions R_i. What we mean by that are the points in π that land in these regions after the pattern 132 has been matched to π.

Now, if the regions R_2 or R_3 contain a point, then there is an occurrence of 132 in π with a larger element corresponding to 3, which contradicts our assumption, so we can assume that R_2 and R_3 are empty. If the largest point in π is contained in R_1, then π contains the pattern p_1. If R_4 is empty, then either R_1 contains the largest point in π, or R_1 is also empty and π contains p_3. If, however, R_4 is not empty, we consider the leftmost point in that region. The following graph shows the occurrence of 132 after we have added a point to the region R_4 (the topmost and rightmost point).



By our previous argument we can consider R_2 and R_3 as shaded, and since we select the leftmost point in R_4, we shade everything to the left of that point in R_4. We then note that if R'_3 contains an inversion, then π contains an occurrence of 132 with a larger element corresponding to the 3, so we can assume that the elements in R'_3 are in descending order. By the same argument, the regions R'_4, R'_5 and R'_6 must be empty.



If R'_2 contains the largest element of π then π contains p_1. We therefore consider the case when R'_2 does not contain the largest element of π. The largest element in π must then either be in R'_3 or be the element we added to R_4. Now if R'_3 is empty, and R'_7 contains an element, then π contains the pattern p_4. Suppose then that R'_7 contains an element, and let c be the rightmost such element, and suppose that R'_3 is non-empty. If c is to the right of the rightmost (and topmost) element in R'_3, then π has an occurrence of p_4. Otherwise, there exist two adjacent elements a and b in R_4 (i.e., R'_3 along with the added point) such that c appears between a and b. Now, if R'_2 contains an element, then π has an occurrence of p_2, otherwise π has an occurrence of p_5. We then consider the case when R'_7 is empty. Now, if either R'_2 or R'_1 contain an element, then π contains the pattern p_2. The only remaining case left to consider is when both R'_1 and R'_2 are empty, in which case π contains p_4.

We have therefore shown that if π contains 132, then $\pi \in \mathsf{Co}(A)$. The cases for 312 and 3241 are analogous.

Thus we have shown that $\mathsf{Co}(A) = \mathsf{Co}(B)$ and therefore the pancake-sortable permutations are

$$P^{-1}(\mathsf{Av}(21)) = \mathsf{Av}(132, 312, 3241). \qquad \square$$

2.6 A linear-time algorithm for pattern avoidance

Let $r : \mathfrak{S}_n \to \mathfrak{S}_n$, $r(\pi) = \pi^r$, denote the reverse of a permutation, and $c : \mathfrak{S}_n \to \mathfrak{S}_n$, $c(\pi) = \pi^c$, denote the complement of a permutation.

Theorem 2.6.1. *A permutation π avoids 4312 if and only if $(S \circ r \circ c \circ Q)(\pi)$ is sorted.*

Proof. We have that $\mathsf{preim}_S(21) = \{231\}$, and that $c(r(231)) = 312$. Our goals is then to find the preimage of $\mathsf{Av}(312)$ under the queue-sort operator, which we did in Example 2.2.3. We therefore we have that

$$(S \circ r \circ c \circ Q)^{-1}(\mathsf{Av}(21)) = Q^{-1}(312) = \mathsf{Av}(4312). \qquad \square$$

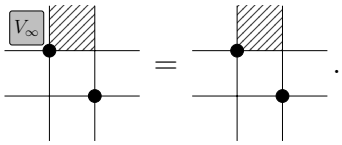
Since all the maps in the composition $S \circ r \circ c \circ Q$ are linear time operators, and it takes linear time to check whether the output is sorted, this provides a linear time algorithm for checking for the avoidance of 4312. Such linear time algorithms have only been known for patterns of length at most 3 as well as the increasing and decreasing patterns of any length ($12\ldots k$ and $k\ldots 21$). Also see Albert et al. [14] for algorithms with running time $n \log n$ for patterns of length 4.

Chapter 3

Preimages of mesh patterns

In this chapter we will extend the algorithm for finding preimages when sorting with a stack of infinite depth, i.e., $\mathsf{preim}_{S_\infty}$, to handle certain types of mesh patterns.

In Section 2.1 we introduced an algorithm preim_{S_d} which was an extension of an algorithm given by Claesson and Úlfarsson [5] for finding preimages of permutation classes under the stack-sort operator (of unlimited depth). Algorithm 3.1 gives a restatement of [5, Algorithm 1], described in terms of meta patterns. This algorithm is obtained by setting $d = \infty$ in preim_{S_d}. Since V_∞ is avoided by all finite permutations, we can omit $\mathcal{C}_2$ from the algorithm, and we can also simplify $\mathcal{C}_3$, and thus we define

$\mathcal{C}_3^\infty =$  $=$.

We then define

$$\mathsf{preim}_S(p) = \bigcup_{\lambda \in \mathsf{cand}_\infty(p)} \mathsf{decorate}_S(p, \lambda),$$

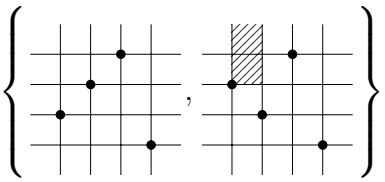
and therefore

$$S^{-1}(\mathsf{Av}(p)) = \mathsf{Av}(\mathsf{preim}_S(p)).$$

As mentioned before, Knuth showed that permutations avoiding the pattern 231 are the permutations sortable with a single pass through a stack. West then classified permutations sortable with two passes through a stack, using *barred patterns*, which can be expressed as mesh patterns with a single shaded box. In general, a permutation $\pi \in \mathfrak{S}_n$, such that $S^k(\pi) = \mathsf{id}_n$, is called *West-k-stack-sortable*.

By applying preim_S to 21, we obtain the preimage of $\mathsf{Av}(21)$ under S, i.e., we obtain a description of the stack-sortable permutations. The algorithm can therefore be used to reproduce Knuth's results. Furthermore, since the stack-sortable permutations are defined by the avoidance of a classical pattern, we can apply preim_S again to 231. By doing so we reproduce West's results and obtain a description of the West-2-stack-sortable permutations, which are

$$\mathsf{preim}_S(231) = \left\{ \right.$$



$$\left. \right\}.$$

Now, since the West-2-stack-sortable permutations are described in terms of mesh patterns, we cannot apply preim_S again to obtain a description of West-3-stack-sortable permutations. However, Úlfarsson, in [3], introduced decorated patterns to give a description of permutations sortable with three passes through a stack. In this chapter we aim to generalize Úlfarsson's approach, and give the preimage of a class of permutations, defined by the avoidance of a certain type of mesh pattern, under the stack-sort operator.

Algorithm 3.1: $\mathsf{decorate}_S$

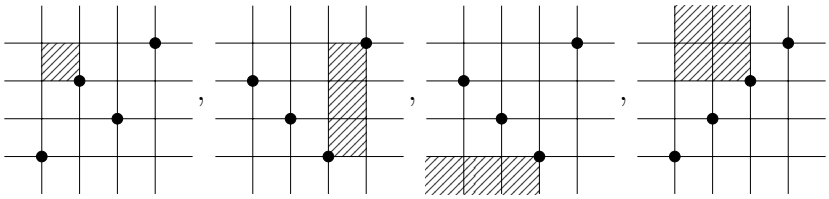
Input: A target pattern p; and $\lambda \in \mathsf{cand}_\infty(p)$

Output: A decorated pattern r describing the preimage of p under S; otherwise, if no such r exists, the execution results in an error

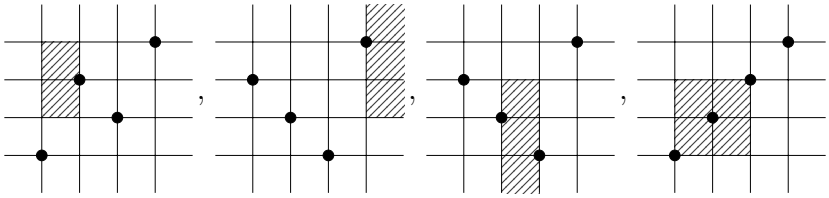
```
1  Let r = (λ, ∅, ∅, ∅, ∅);
2  for v ∈ inv(λ) do
3      if v ∈ inv(p) then
4          if C_1 is applicable to v in r then
5              apply C_1 to v in r;
6          else
7              error;
8      else
9          if C_3^∞ is applicable to v in r then
10             apply C_3^∞ to v in r;
11         else
12             error;
13 return r;
```

3.1 Finding preimages of mesh patterns

We will now introduce an algorithm, $\mathsf{meshpreim}_S$, that finds the preimage of a class of permutations, defined by the avoidance of a certain type of mesh pattern, under the stack-sort operator. The domain of $\mathsf{meshpreim}_S$, i.e., the mesh patterns whose preimage we can determine, denoted by $\mathfrak{M}$, is defined by the meta patterns $\mathcal{M}_i$, given in Table 3.1. More precisely, a mesh pattern is in $\mathfrak{M}$ if it is the result of applying $\mathcal{M}_i$ to a permutation. For example, the patterns



are in $\mathfrak{M}$, whereas



are not.

Similar to the previously defined preimage algorithms, $\mathsf{meshpreim}_S$ proceeds in two steps. In the first step we find candidates by applying preim_S to the underlying classical pattern of the target permutation. In the second step, $\mathsf{mdecorate}_S$, we apply meta patterns to the candidates to obtain the patterns that describe the preimage. We must consider 10 cases, corresponding to each of the meta patterns $\mathcal{M}_1, \ldots, \mathcal{M}_{10}$ that can be applied to a permutation to obtain the target mesh pattern.

In $\mathsf{mdecorate}_S$ we use the fact that if M is a set of decorated patterns, and p is a mesh pattern, then if $S^{-1}(\mathsf{Av}(p)) = \mathsf{Av}(M)$ it follows that $S^{-1}(\mathsf{Co}(p)) = \mathsf{Co}(M)$. This holds since for any permutation π we have that $\pi \in \mathsf{Av}(M)$ if and only if $\pi \notin \mathsf{Co}(M)$.

For each candidate pattern q we obtain for a target pattern p, it holds that if a permutation π contains q then $S(\pi)$ contains the underlying classical pattern of p. What we therefore do in the second step, $\mathsf{mdecorate}_S$, is to add restrictions to q to guarantee that π contains q, only if $S(\pi)$ contains p.

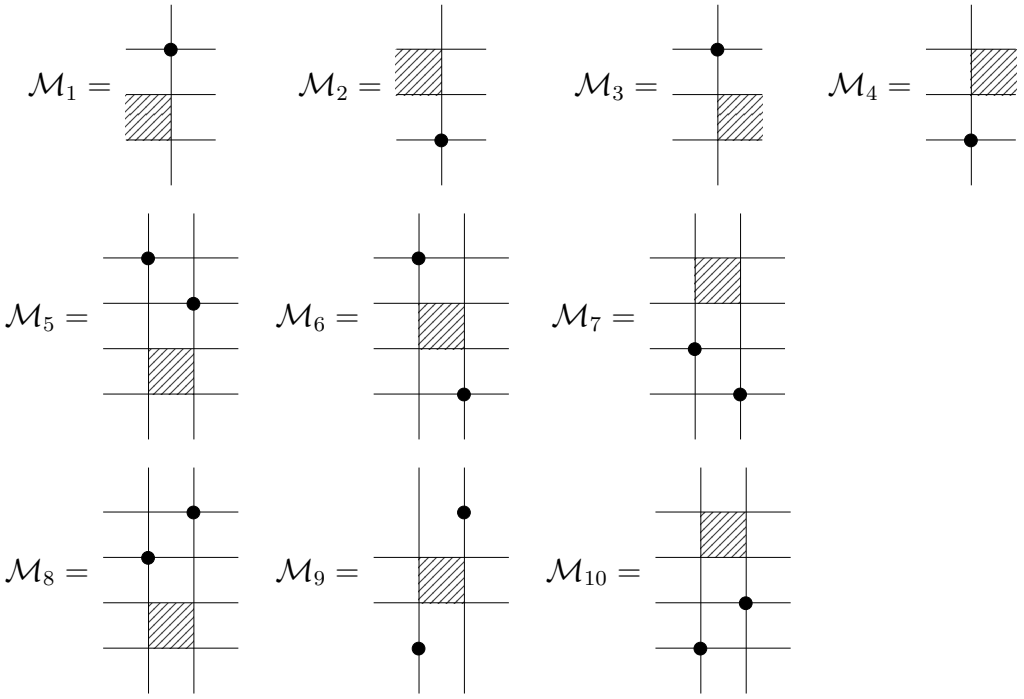
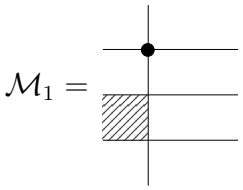
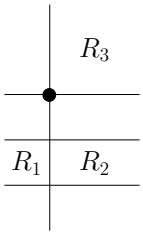


Figure 3.1: The meta patterns describing the domain of $\mathsf{meshpreim}_S$

To demonstrate this, let us consider the first case, the case when the target pattern was obtained by applying



to a permutation. Let p be the target pattern and π be the underlying classical pattern of p. In the first step of $\mathsf{meshpreim}_S$, i.e., $\mathsf{preim}_S(\pi)$, we obtain a set of patterns all of which will have the following layout.



The point in the layout, a, refers to the point to which $\mathcal{M}_1$ was applied in π to obtain p, and the additional horizontal lines refer to the upper and lower bounds of the shaded region in p. Note that after preim_S has been applied to π, the position of the point may change from π, but the upper and lower bounds of the shaded region will remain the same.

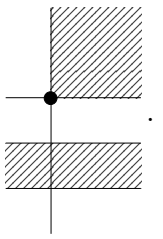
Our aim is now to show how shadings, markings and decorations can be added to the pattern to make sure that the shaded region in $\mathcal{M}_1$ will be empty after a permutation containing this pattern passes through a stack. What we mean by that is if a permutation λ contains the pattern described by the layout, we know that it will contain π after a pass through a stack. Our aim is that, for all occurrences of π in $S(\lambda)$ (i.e. λ after a pass through the stack), it holds that the shaded region of p will be empty. I.e., we want to make sure that $S(\lambda)$ contains p.

For the sake of brevity in our argument, we will refer to points in regions of the layout or meta pattern. What we mean by that are the elements of any permutation λ containing the pattern, that end up in the specified region of the pattern after the graph of the pattern has been matched to the graph of λ.

Now, going back to our example, what we do not want is for a point to end up in the shaded region, R', of $\mathcal{M}_1$, after a pass through the stack. First of all, we note that if a point is contained in R_1, then it will be pushed onto the stack, and then subsequently popped off by a (or possibly before), and therefore it will end up in R'. The region R_1 must therefore be shaded in the layout.

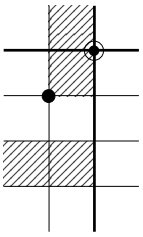
Now, suppose we have an element b in R_2. Since b is smaller than a, it can land on top of a in the stack, and come out before a and therefore land in region R'. However, if an element c, larger than a (i.e., c is in region R_3), appears before b, then c will pop a off the stack, before b arrives at the stack, and therefore b will not land in R'. We must therefore consider two cases.

1. The region R_3 is empty. In this case, R_2 must also be empty, since if R_2 contains an element, then it will land above a on the stack and be popped off the stack before a, and therefore it will land in R'. This case is described by the meta pattern

$\mathcal{G}_{1,1} =$  .

2. The region R_3 contains an element c. Without loss of generality, we can assume that c is the leftmost element in R_3. We then note that, if an element b appears in R_2 between a and c, then b will land above a in the stack. Thus b will be popped off the stack before a and will therefore land in R'. The region R_2 must therefore

be empty up to the point when c appears. This case is described by the meta pattern

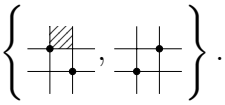
$\mathcal{G}_{1,2} =$  .

The circled point corresponds to the element c. We shade everything in R_3 to the left of c to choose c as the leftmost element in R_3.

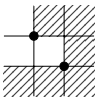
Example 3.1.1. Suppose we want to find $\mathsf{meshpreim}_S(p)$, where

$p =$  ,

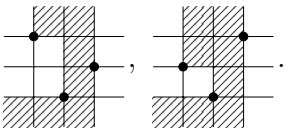
which is obtained by applying $\mathcal{M}_1$ to 1, π_0^h and π_1^h in $\pi = 12$. In the first step we obtain

$\mathsf{preim}_S(12) = \{q, r\} =$  .

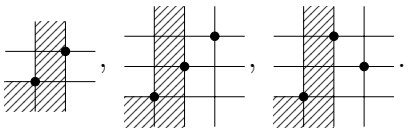
Applying the meta pattern $\mathcal{G}_{1,1}$ to 1, q_0^h and q_1^h in q we get



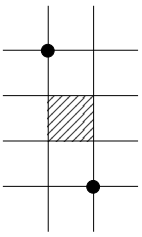
and applying $\mathcal{G}_{1,2}$ we obtain

 .

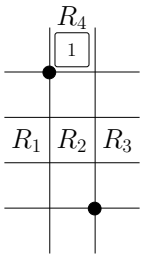
Applying $\mathcal{G}_{1,2}$ to 1, r_0^h and r_1^h in r we get

 .

Let us now consider the case when the target pattern was obtained by applying

$\mathcal{M}_6 =$ 

to a permutation. Let p be the target pattern and π be the underlying classical pattern of p. In the first step of $\mathsf{meshpreim}_S$, i.e., $\mathsf{preim}_S(\pi)$, we obtain a set of patterns all of which will have the following layout (after $\mathcal{C}_1$ has been applied to the inversion in π).



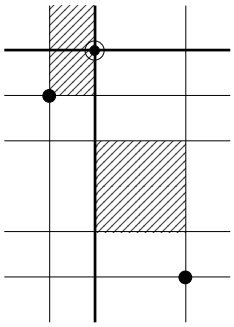
We let R' denote the shaded region of $\mathcal{M}_6$ and let a and b denote the left and right points in the layout, respectively. Our aim is, again, to add restrictions to the layout such that R' will be empty after the permutation matching the layout pattern passes through the stack.

If R_1 contains an element, that element will be popped off the stack by a (or even before that), and will therefore appear before a in the output, and therefore not land in R'. The candidate pattern requires there to be an element in R_4, so a will be popped off the stack before b appears. If an element is present in R_3 then that element (or a previous element) will pop b off the stack, and therefore appear after b in the output and not land in R'. We therefore need only consider elements in R_2.

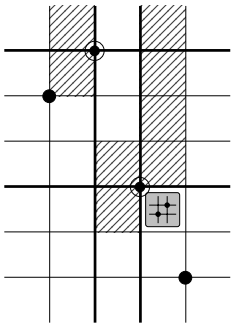
First, we observe that the pattern requires there to be an element in R_4. Let c be the leftmost element in R_4. Now, if an element in R_2 appears before c, it will either end above a in the stack, or be popped off the stack before c appears. The element a, however, will not leave the stack until c appears, since it is the leftmost element larger than a. Thus an element in R_2 that appears before c cannot land in R'.

Now, after c appears, a has been popped off the stack. Therefore, if an element appears in R_2, after c, and is subsequently popped off the stack, before b appears, it will end in R'. We must then consider two cases.

1. The region R_2 is empty after c appears, in which case no element will land in R'. This case is described by the meta pattern

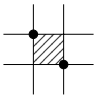
$\mathcal{G}_{6,1} =$  .

2. There is an element d in R_2 that appears after c. Without loss of generality, we can assume that d is the leftmost such element. Now, if there is an element larger than d that appears after d, in R_2, then d will be popped off the stack, and it will land in R'. We must therefore forbid all larger elements than d after d and before b. There can, however, appear an element smaller than d, after d, in R_2. That element will land on top of d in the stack, and if it is not popped off the stack before b appears, b will go on top of the stack, and therefore appear before the two elements in the output. Any number of elements can, in fact, come after d in R_2, as long as they are in decreasing order. If there is a non-inversion after d in R_2, it will cause the stack to be popped, and an element will land in R'. This case is described by the meta pattern

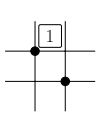
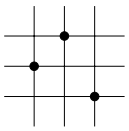
$\mathcal{G}_{6,2} =$  .

In this pattern the left circled point represents c and the right circled point represents d. We add a shading after and above d to forbid all elements larger than d before b. We also add an avoidance decoration after and below d to denote a sequence of decreasing elements.

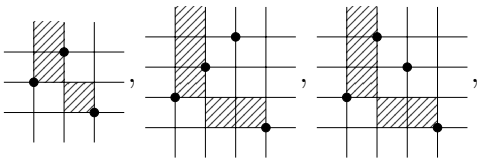
Example 3.1.2. Suppose we want to find $\mathsf{meshpreim}_S(p)$, where

$p =$  ,

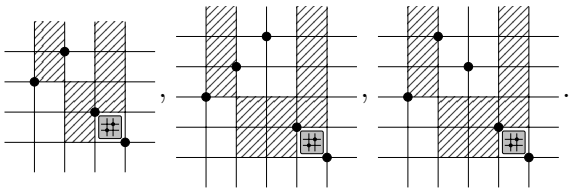
which is obtained by applying $\mathcal{M}_6$ to 21, π_1^h and π_2^h in $\pi = 21$. In the first step we obtain

$\mathsf{preim}_S(21) = \{q\} =$  $=$  .

Applying the meta pattern $\mathcal{G}_{6,1}$ to 21, q_1^h and q_2^h in q we get



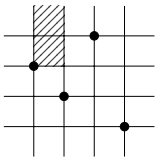
and applying $\mathcal{G}_{6,2}$ we obtain



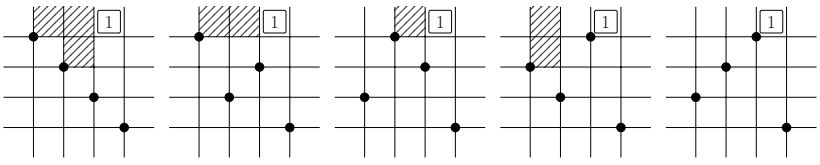
A complete list of meta patterns, applied for each of the 10 cases, is given in Appendix B.

3.2 West-3-stack-sortable permutations

Let $w_1 = 2341$ and

$w_2 =$  .

We now have all the tools necessary to determine the West-3-stack-sortable permutations, which are $S^{-1}(\mathsf{Av}(w_1, w_2))$. The preimage of $\mathsf{Av}(w_1)$, given by $\mathsf{preim}_S(2341)$, is defined by the avoidance of the following patterns.



The preimage of $\mathsf{Av}(w_2)$, given by $\mathsf{meshpreim}_S(w_2)$, is defined by the patterns given in Table 3.1. Note that patterns containing 23451 (which is in $\mathsf{Av}(w_1)$) have been removed since they are redundant. The West-3-stack-sortable permutations are therefore defined by the avoidance of the union of these two sets of patterns.

We have now given algorithms that describe stack-sortable permutations, West-2-stack-sortable permutations and West-3-stack-sortable permutations, and therefore completely automated the proofs of the discoveries made by Knuth, West and Úlfarsson.

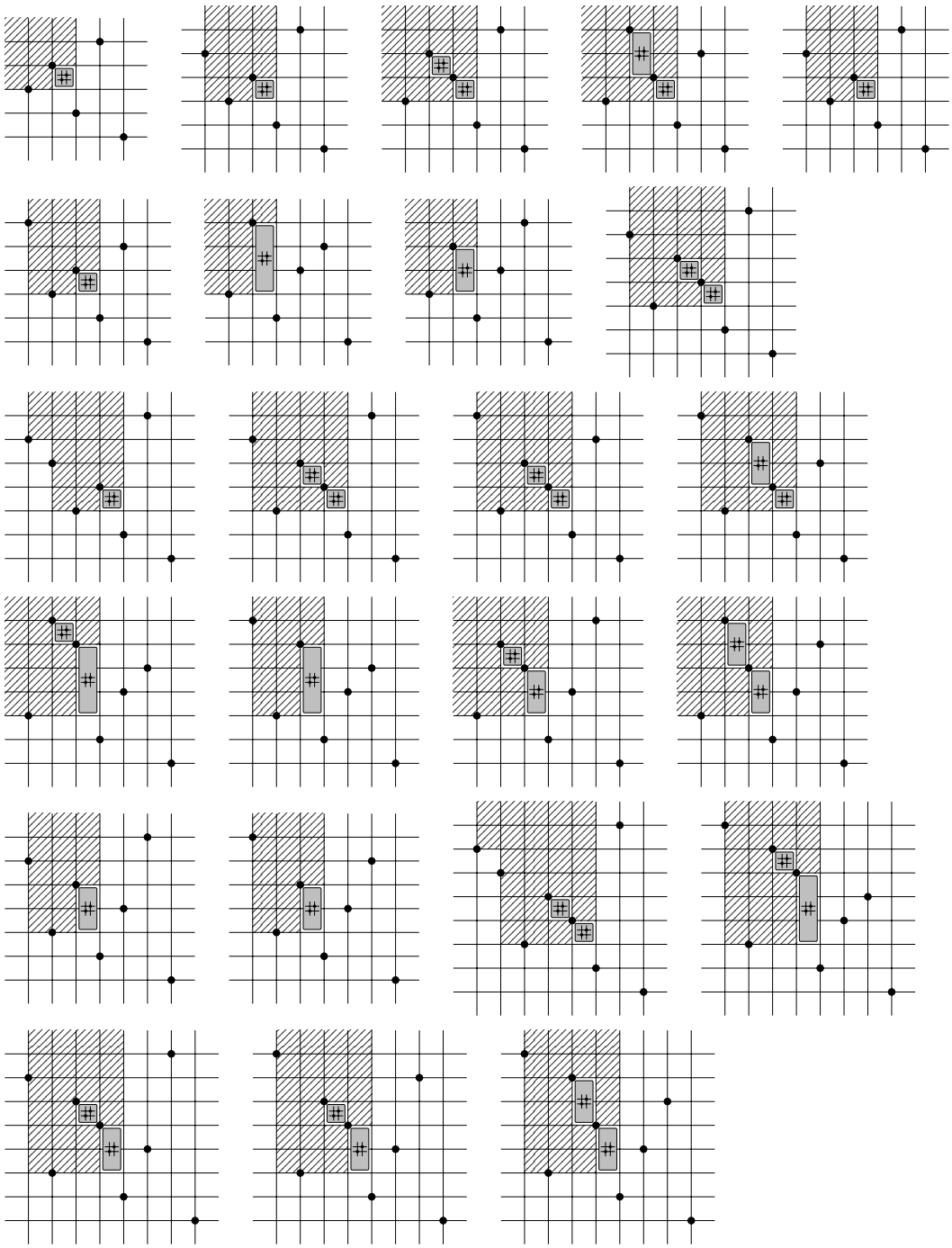


Table 3.1: The patterns defining the preimage of w_2 under S

Chapter 4

Conclusions

In this thesis we give a general method for producing algorithms for describing preimages of sorting operators. First we describe the cases when inversions in a candidate pattern become inversions and non-inversions in a target pattern, in terms of meta patterns. We can then give an algorithm to generate patterns that describe the preimage of the avoidance class of the target. The process of finding the meta patterns, however, is done by hand. It should be possible to, at least partially, automate this process. Given a sorting operator f with the property that $\mathsf{inv}(f(\pi)) \subseteq \mathsf{inv}(\pi)$, we believe it should be possible to give an algorithm that conjectures meta patterns that describe inversions that become inversions and non-inversions. This could even be possible if we relax the condition that the operator does not "create" inversions, such as in pancake sort.

Another possible extension to our work is finding preimage algorithms for general mappings, not just sorting operators. For the symmetries, i.e., reverse, inverse and complement, this is trivial, since we can define the reverse, complement and inverse of decorated patterns. In general, for non-trivial mappings, finding candidates might become difficult, and our approach of determining what happens to inversions and non-inversions, will in general not be suitable. An entirely new approach might therefore be necessary.

We also suggest the following extension of $\mathsf{meshpreim}_S$ to handle any mesh pattern target, not just the elements of $\mathfrak{M}$. We note that the shadings of any mesh pattern p can be split up into smaller shadings, $s_1, s_2, \ldots, s_k$, such that each of the smaller shadings is the result of applying some $\mathcal{M}_i$ to the underlying classical pattern of p. We can then apply $\mathsf{meshpreim}_S$ to each candidate, as if p only had the shading s_1. We then apply $\mathsf{meshpreim}_S$ again, to the patterns obtained, as if p only had the shading s_2, and so forth, for each s_i.

The final extension we propose is determining what type of pattern definitions are closed under the preimage of the stack-sort operator (or any other sorting operator). We believe

that the pattern definitions, described by the avoidance of a finite number of decorated patterns, are not closed under the stack-sort operator. In particular, we believe that decorated patterns are not expressive enough to describe, with a finite number of patterns, the West-4-stack-sortable permutations.

Bibliography

[1] D. E. Knuth, *The Art of Computer Programming*. Addison-Wesley Publishing Co., Reading, Mass.-London-Amsterdam, third ed., 1997. Volume 1: Fundamental algorithms, Addison-Wesley Series in Computer Science and Information Processing.

[2] J. West, *Permutations with forbidden subsequences and stack-sortable permutations*. PhD thesis, MIT, 1990.

[3] H. Ulfarsson, "Describing West-3-stack-sortable permutations with permutation patterns," *Electron. J. Combin.*, vol. 67, pp. Article B67d, 20 pp. (electronic), 2012.

[4] P. Brändén and A. Claesson, "Mesh patterns and the expansion of permutation statistics as sums of permutation patterns," *Electron. J. Combin.*, vol. 18, 2011.

[5] 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, pp. 595–606, Assoc. Discrete Math. Theor. Comput. Sci., Nancy, 2012.

[6] H. Ulfarsson, "A unification of permutation patterns related to Schubert varieties," *Pure Math. Appl. (PU.M.A.)*, vol. 22, no. 2, pp. 273–296, 2011.

[7] M. Bona, "A survey of stack-sorting disciplines," *Electron. J. Combin*, vol. 9, p. 1, 2003.

[8] 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.*, vol. 2, no. 2, pp. 231–243, 2011.

[9] 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.

[10] D. Avis and M. Newborn, "On pop-stacks in series," *Utilitas Mathematica*, vol. 19, pp. 129–140, 1981.

[11] W. H. Gates and C. H. Papadimitriou, "Bounds for sorting by prefix reversal," *Discrete Mathematics*, vol. 27, no. 1, pp. 47 – 57, 1979.

[12] B. Chitturi, W. Fahle, Z. Meng, L. Morales, C. O. Shields, I. H. Sudborough, and W. Voit, "An (18/11)n upper bound for sorting by prefix reversals," *Theor. Comput. Sci.*, vol. 410, pp. 3372–3390, Aug. 2009.

[13] L. Bulteau, G. Fertin, and I. Rusu, "Pancake flipping is hard," *CoRR*, vol. abs/1111.0434, 2011.

[14] M. H. Albert, R. E. L. Aldred, M. D. Atkinson, and D. A. Holton, "Algorithms for pattern involvement in permutations," in *Algorithms and computation (Christchurch, 2001)*, vol. 2223 of *Lecture Notes in Comput. Sci.*, pp. 355–366, Berlin: Springer, 2001.

[15] W. Stein *et al.*, *Sage Mathematics Software (Version 5.8)*. The Sage Development Team, 2013. `http://www.sagemath.org`.

Appendix A

Implementation of preimage algorithms

A.1 Implementation in Sage

All the algorithms discussed in this thesis, except preim_P, have been implemented in the computer algebra system Sage [15]. The code can be obtained from `https://bitbucket.org/hjaltim/thesis` and is split into the following files.

`pattern_classes.sage`
: Contains implementations of classical patterns, mesh patterns, marked mesh patterns and decorated patterns as Sage classes.

`preim.sage`
: Contains implementations of preim_{S_d}, preim_Q, preim_T and preim_I.

`meshpreim.sage`
: Contain implementations of preim_S and $\mathsf{meshpreim}_S$.

`sorting_functions.sage`
: Contain implementations of the sorting operators S, S_d, Q, T, I and P.

`preim_tests.sage`
: Contain correctness tests for the implementations of preim_{S_d}, preim_Q, preim_T and preim_I.

`meshpreim_tests.sage`
: Contain correctness tests for the implementations of $\mathsf{meshpreim}_S$.

The installation instructions for Sage can be found on the project's website. After starting Sage (typing `sage` in a terminal) you can load the necessary code to run the preimage algorithms with the following commands.

```
sage: load 'pattern_classes.sage'
sage: load 'preim.sage'
sage: load 'meshpreim.sage'
```

The set $\mathsf{preim}_{S_3}(132)$ can be obtained by the following command.

```
sage: preimSd(3,[1,3,2])
```

The set $\mathsf{preim}_{Q}(312)$ can similarly be obtained by the following command.

```
sage: preimQ([3,1,2])
```

The patterns defining the West-2-stack-sortable permutations can be obtained as follows.

```
sage: preimS([1,3,2])
```

The patterns defining the West-3-stack-sortable permutations can then be obtained as follows.

```
sage: west2 = preimS([1,3,2])
sage: exp_west2 = sum(map(lambda x: x.expand(),west2),[])
sage: sum(map(meshpreimS,exp_west2),[])
```

To visualize a pattern, we can call the `show` function. For example, to visualize the mesh pattern $(132, \{(1,3),(2,3)\})$, we can use the following command.

```
sage: p = MeshPattern([1,3,2],[(1,3),(2,3)])
sage: show(p)
```

To visualize a list (or set) of patterns, we can use the function `show_multiple`. For example, we can visualize the set $\mathsf{preim}_{Q}(312)$ as follows.

```
sage: s = preimQ([3,1,2])
sage: show_multiple(s,3)
```

A.2 Sorting with a stack of depth d

Algorithm A.1 gives a low-level description of preim_{S_d} without the use of meta patterns.

Algorithm A.1: $\mathsf{decorate}_{S_d}$ (without meta patterns)

Input: The depth d of the stack; a pattern p; and $\lambda \in \mathsf{cand}_d(p)$
Output: A (possibly empty) set of decorated patterns T

Let $T = \{(\lambda, \varnothing, \varnothing, \varnothing, \varnothing)\}$ and let n be the length of λ;
for $(i, j) \in \mathsf{inv}(\lambda)$ **do**
  Let $R_1 = [\![\lambda^{-1}(i), \lambda^{-1}(j) - 1]\!] \times [\![i, n]\!]$;
  Let $R_2 = [\![0, \lambda^{-1}(i) - 1]\!] \times [\![i, n]\!]$;
  for $r = (\lambda, S, M, D, C) \in T$ **do**
    Remove r from T;
    if $(i, j) \in \mathsf{inv}(p)$ **then**
      if $R_1 \not\subseteq S$ **then**
        if *An element of* λ *is contained in* R_1 **then**
          Add r to T;
        else
          Add $(\lambda, S, M \cup \{(R_1 \setminus S, 1)\}, D, C)$ to T;
      if *No element of* λ*, no marking in* M*, or decoration in* C*, is contained in* $R_1 \cup S$ *and* R_2 *is not contained in any decoration in* D **then**
        Add $(\lambda, S \cup R_1, M, D, C \cup \{(R_2, V_{d-1})\})$ to T;
    else if *No element of* λ*, no marking in* M*, or decoration in* C*, is contained in* $R_1 \cup S$ *and no decoration from* C *is contained in* R_2 **then**
      Add $(\lambda, S \cup R_1, M, D \cup \{(R_2, V_{d-1})\}, C)$ to T;

Appendix B

Meta patterns for $\mathsf{meshpreim}_S$

Tables B.1–B.10 contain the meta patterns applied in each of the 10 cases in $\mathsf{meshpreim}_S$. Where applicable, the dotted line refers to the rightmost element larger than the upper bound of the shaded region of the target, or the beginning of the pattern, if no such element exists. Note that Case 4, shown in Table B.4, is split into two sub-cases. The first case applies when the shaded region in the target does not reach the top of the pattern, and the second case applies when the shaded region does reach the top of the pattern.

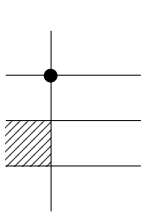
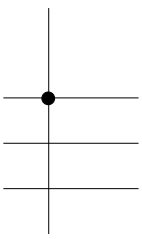
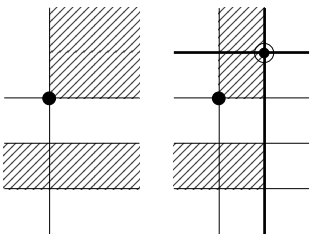
Target	**Output from** preim_S	**Meta patterns applied in** $\mathsf{meshpreim}_S$
		

Table B.1: Meta patterns for $\mathsf{meshpreim}_S$ – Case 1

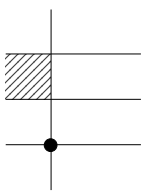
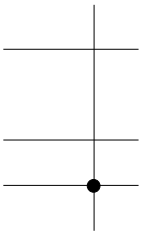
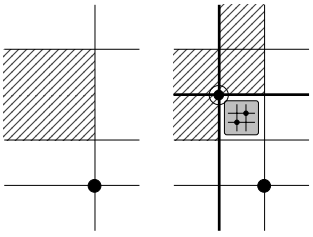
Target	**Output from** preim_S	**Meta patterns applied in** $\mathsf{meshpreim}_S$
		

Table B.2: Meta patterns for $\mathsf{meshpreim}_S$ – Case 2

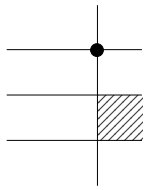
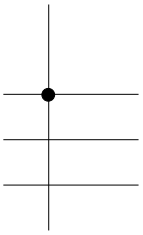
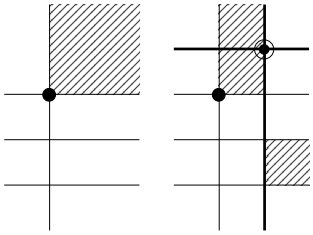
Target	**Output from** preim_S	**Meta patterns applied in** $\mathsf{meshpreim}_S$
		

Table B.3: Meta patterns for $\mathsf{meshpreim}_S$ – Case 3

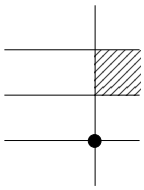
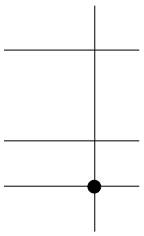
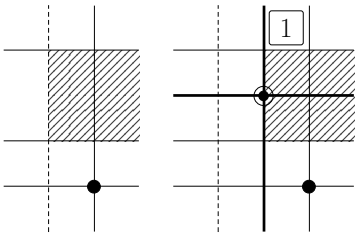
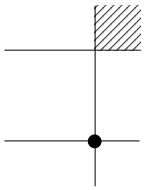
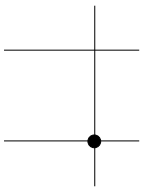
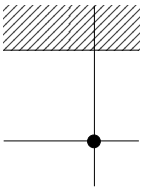
Target	**Output from** preim_S	**Meta patterns applied in** $\mathsf{meshpreim}_S$
		
		

Table B.4: Meta patterns for $\mathsf{meshpreim}_S$ – Case 4

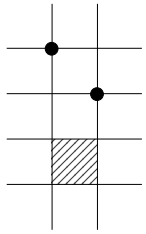
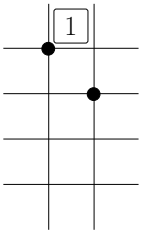
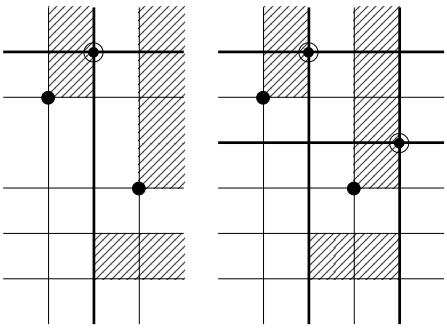
Target	**Output from** preim_S	**Meta patterns applied in** $\mathsf{meshpreim}_S$
		

Table B.5: Meta patterns for $\mathsf{meshpreim}_S$ – Case 5

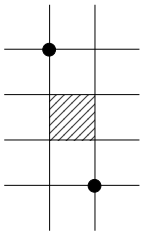
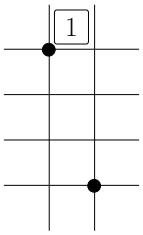
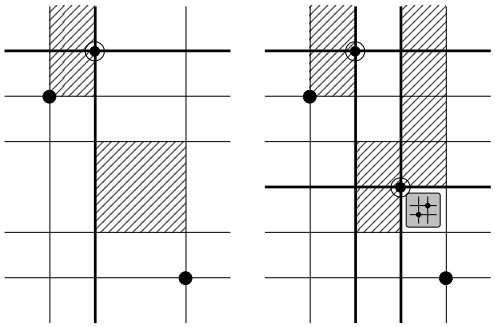
Target	**Output from** preim$_S$	**Meta patterns applied in** meshpreim$_S$
		

Table B.6: Meta patterns for meshpreim$_S$ – Case 6

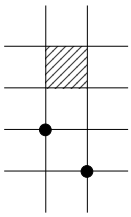
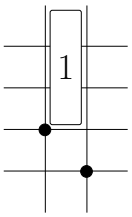
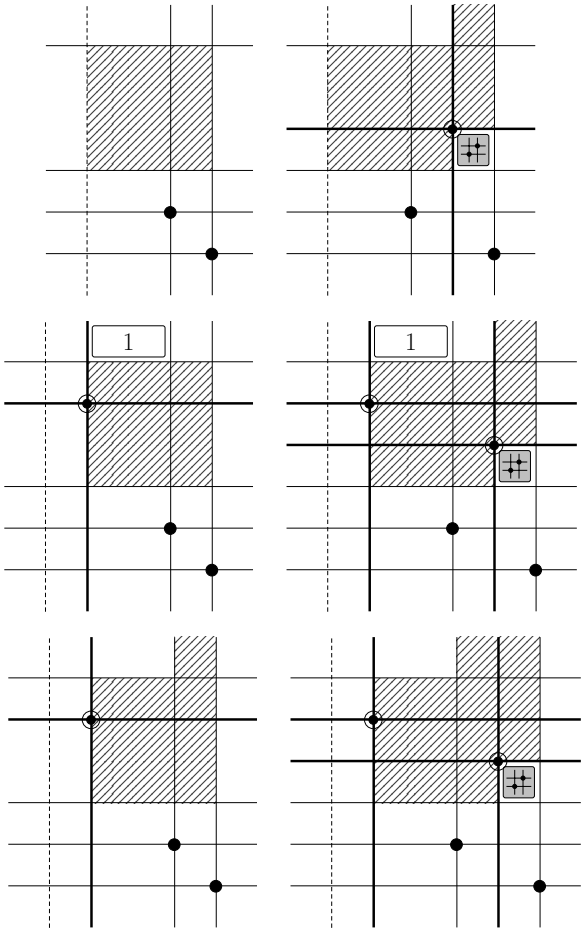
Target	**Output from** preim$_S$	**Meta patterns applied in** meshpreim$_S$
		

Table B.7: Meta patterns for meshpreim$_S$ – Case 7

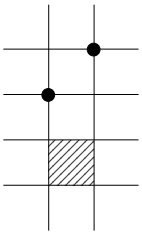
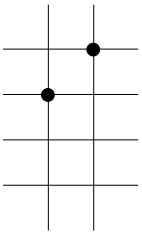
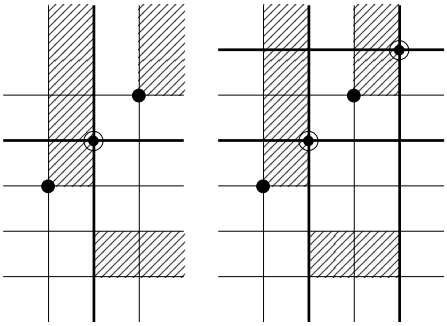
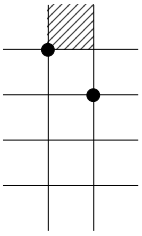
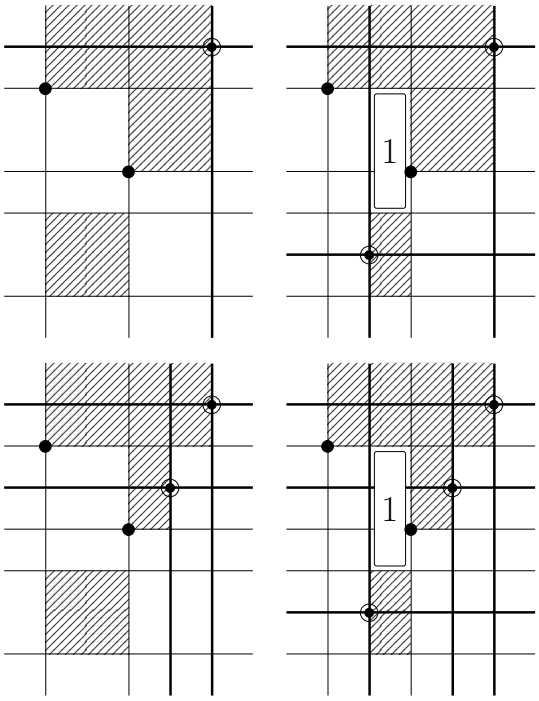
Target	**Output from** preim_S	**Meta patterns applied in** $\mathsf{meshpreim}_S$
		
		

Table B.8: Meta patterns for $\mathsf{meshpreim}_S$ – Case 8

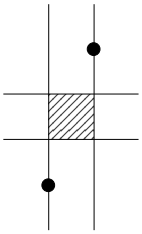
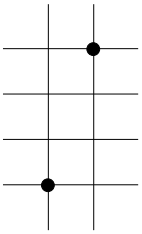
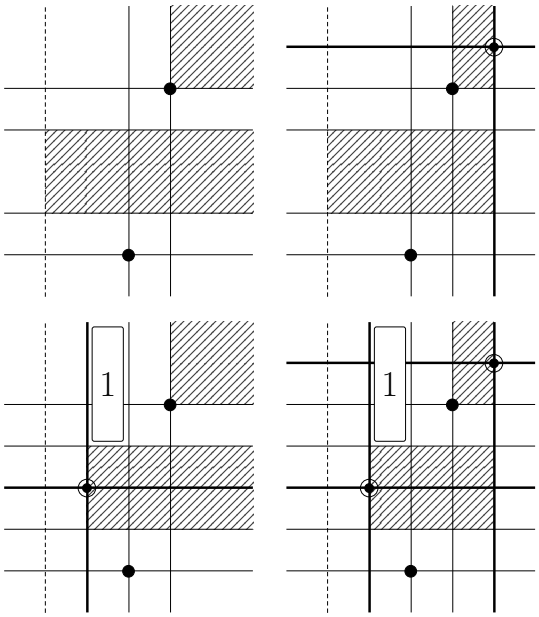
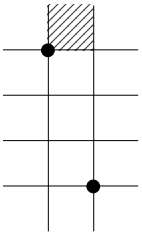
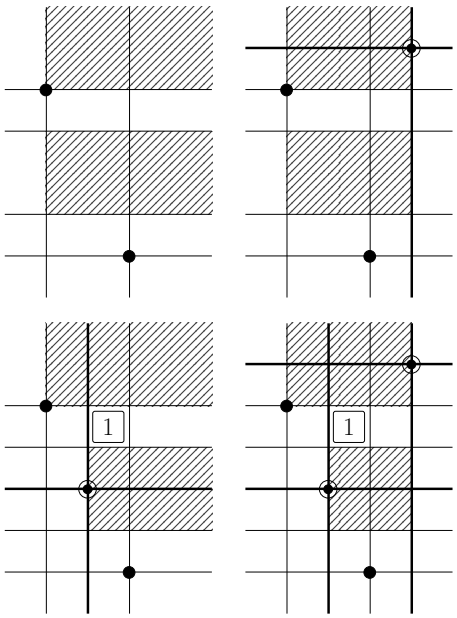
Target	**Output from** preim_S	**Meta patterns applied in** $\mathsf{meshpreim}_S$
		
		

Table B.9: Meta patterns for $\mathsf{meshpreim}_S$ – Case 9

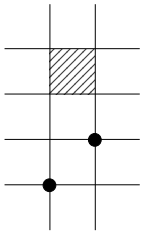
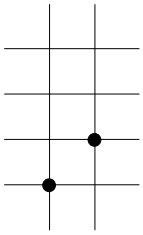
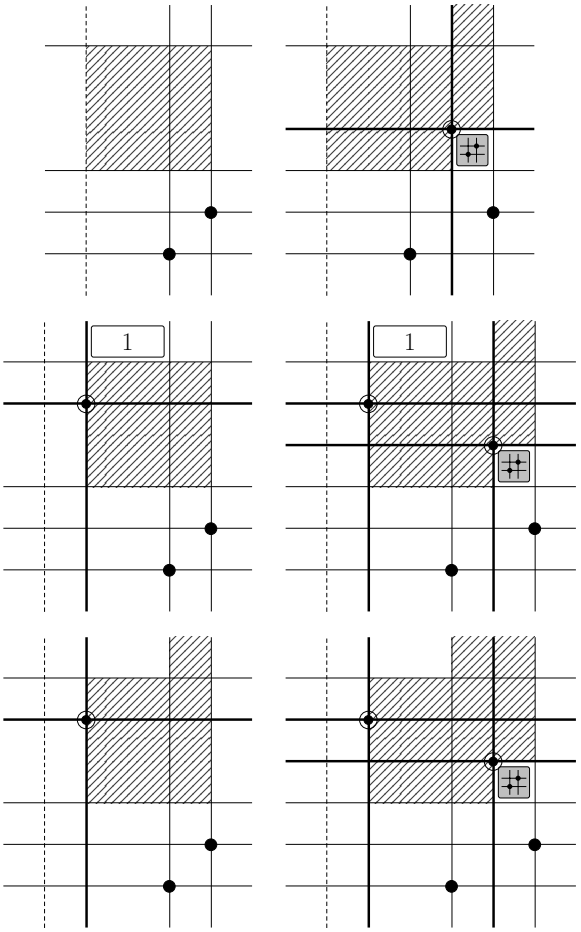
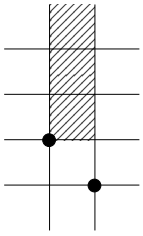
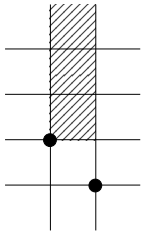
Target	**Output from** preim_S	**Meta patterns applied in** $\mathsf{meshpreim}_S$
		
		

Table B.10: Meta patterns for $\mathsf{meshpreim}_S$ – Case 10

School of Computer Science
Reykjavík University
Menntavegi 1
101 Reykjavík, Iceland
Tel. +354 599 6200
Fax +354 599 6201
www.reykjavikuniversity.is
ISSN 1670-8539