7-2018

Generating Permutations with Restricted Containers

Michael H. Albert
University of Otago

Cheyne Homberger
University of Maryland - Baltimore County

Jay Pantone
Marquette University, jay.pantone@marquette.edu

Nathaniel Shar
Rutgers University

Vincent Vatter
University of Florida

Journal of Combinatorial Theory, Series A, Vol. 157 (July 2018): 205-232. DOI.

Generating Permutations with Restricted Containers

Michael H. Albert
Department of Computer Science, University of Otago, Dunedin, New Zealand
Cheyne Homberger
Department of Mathematics, University of Maryland Baltimore County, Baltimore, MD
Jay Pantone[1]
Department of Mathematics, Dartmouth College, Hanover, NH
Nathaniel Shar
Department of Mathematics, Rutgers University, New Brunswick, NJ
Vincent Vatter[1]
Department of Mathematics, University of Florida, Gainesville, FL

Abstract

We investigate a generalization of stacks that we call $\mathcal{C}$- machines. We show how this viewpoint rapidly leads to functional equations for the classes of permutations that $\mathcal{C}$-machines generate, and how these systems of functional equations can be iterated and sometimes solved. General results about the rationality, algebraicity,

and the existence of Wilfian formulas for some classes generated by $\mathcal{C}$-machines are given. We also draw attention to some relatively small permutation classes which, although we can generate thousands of terms of their counting sequences, seem to not have D-finite generating functions.

Keywords

Permutation patterns, Enumeration, Stack, Sorting machine

1. Introduction

1.1. History and context

The study of permutation patterns is generally considered to have been started by Knuth, when he proved in the first volume of *The Art of Computer Programming* [23, Section 2.2.1] that a permutation can be generated by a stack if and only if it avoids 312 (i.e., does not contain three entries in the same relative order as 312). That initial work was followed by a series of papers dealing with permutations sorted or generated by machines of varying kinds. Prompted apparently by an observation of Pratt [30], this led to the general consideration of *permutation classes*, which has become a highly active area of combinatorial research in its own right. We refer to the last author's survey [35] for a broad overview of this area.

Many works in this area have focused on enumerative questions about permutation classes, as we do here. We would like to emphasize however that the ability to effectively enumerate a class is really a proxy indicating that we understand its structure at some sufficient level of detail—one of the main goals of research in the area is to refine our understanding of such structure.

In this work we look back to the structure of permutations generated by machines, where a machine is simply a container subject to certain restrictions (analogous to the stack in Knuth's original work). We show that this notion captures the equivalence between various permutation classes, specifically most of those enumerated by the Catalan or Schröder numbers. We are able to provide rigorous formulas that enumerate some other permutation classes of this type. Finally, because of the simplicity of the underlying mechanism we are able to compute a large number of initial terms in the generating functions of some other, seemingly equally uncomplicated, permutation classes and establish by empirical methods that their generating functions appear not to be D-finite.

1.2. Concepts and definitions

We are solely concerned with classical permutation patterns, in which the permutation π *contains* the permutation σ if π contains a subsequence *order isomorphic* (i.e., with the same pairwise comparisons) to σ. Otherwise, π *avoids* σ. For example, 53412 contains 321, as evidenced by any of the subsequences 531, 532, 541, or 542. The containment relation is a partial order, and a *permutation class* is a downset, or lower order ideal, of permutations under this order.

As with any downset in a poset, every permutation class can be described as

$$\mathrm{Av}(B) = \{\pi : \pi \text{ avoids all } \beta \in B\}$$

for some set B of permutations. We may take the set B to be an *antichain*, i.e., a set of pairwise incomparable permutations, and if B is an antichain its choice is unique, and we refer to it as the *basis* of the class in question. Given a permutation class $\mathcal{C}$ and nonnegative integer n, we denote by $\mathcal{C}_n$ the set of permutations in $\mathcal{C}$ of length n, and refer to

$$\sum_{n\geq 1} |\mathcal{C}_n| x^n = \sum_{\pi\in\mathcal{C}} x^{|\pi|}$$

as its generating function. Two classes are *Wilf-equivalent* if they have the same generating function. The *growth rate* of the permutation class $\mathcal{C}$ is defined as

$$\mathrm{gr}(\mathcal{C}) = \lim_{n\to\infty} \sqrt[n]{|\mathcal{C}_n|}$$

when this limit exists.

Some permutation classes are trivially Wilf-equivalent via the *symmetries* of the permutation order. Given a permutation $\pi = \pi(1)\pi(2)\cdots\pi(n)$, the reverse of π is the permutation π^{r} defined by $\pi^{\mathrm{r}}(i) = \pi(n+1-i)$, the complement of π is the permutation π^{c} defined by $\pi^{\mathrm{c}}(i) = n+1-\pi(i)$, and the (group-theoretic) inverse of π is the permutation π^{-1} defined by $\pi^{-1}(\pi(i)) = \pi(\pi^{-1}(i)) = i$. From the geometric viewpoint, reversing a permutation consists of reflecting its plot over any vertical line, complementing it consists of reflecting its plot over any horizontal line, and inverting it consists of reflecting its plot over a line of slope 1.

We need to define the two operations on permutations illustrated in Fig. 1. Given permutations π of length k and σ of length ℓ, their *(direct) sum* is defined as

$$(\pi \oplus \sigma)(i) = \begin{cases} \pi(i) & \text{for } 1 \le i \le k, \\ \sigma(i-k)+k & \text{for } k+1 \le i \le k+\ell. \end{cases}$$

The analogous operation depicted on the right of Fig. 1 is called the *skew sum*. We can now characterize the class of permutations which can be generated by a $\mathcal{C}$-machine.

Fig. 1. The sum and skew sum operations.

We are concerned here with a fairly general family of machines. Suppose that $\mathcal{C}$ is a permutation class. A $\mathcal{C}$-machine is a machine consisting of a container that holds partial permutations. In using this $\mathcal{C}$-machine to generate permutations from the input $12\cdots n$ we may at any time perform one of three operations:

- remove the next entry from the input and immediately append it to the end of the output (a *bypass*),
- remove the next entry from the input and place it anywhere in the container in such a way that the partial permutation in the container is in the same relative order as a permutation in the class $\mathcal{C}$ (a *push*), or
- remove the leftmost entry from the container and append it to the end of the output (a *pop*).

The operation of this machine could be analogized to the situation of an administrator who, upon receiving a new task, may choose either to perform it immediately (the bypass option) or file it away. The administrator may also, of course, choose to perform some of the filed tasks, but only in the order in which they lie in the filing cabinet, and the possible orderings of the tasks within the filing cabinet is restricted.

We refer to a sequence of operations of this form as a *generation sequence* for the permutation π that is eventually produced. Formally, a generation sequence corresponds to a sequence of letters specifying which of these three actions was taken and in the case of a push operation, where the new element was pushed.

For a simple example, consider the $\mathrm{Av}(12)$-machine, illustrated on the left of Fig. 2. In this machine the container may only contain entries in decreasing order. Thus in generating permutations with the $\mathrm{Av}(12)$-machine, if we push an entry from the input to the container we must place it at the leftmost end of the container (because at any point in time all entries in the input are necessarily greater than every entry in the container). We may also pop from the beginning of the container. In this machine (but not in general) a bypass is

equivalent to a push followed immediately by a pop, and therefore we may ignore bypasses. Thus the $\mathrm{Av}(12)$-machine is equivalent to a stack.

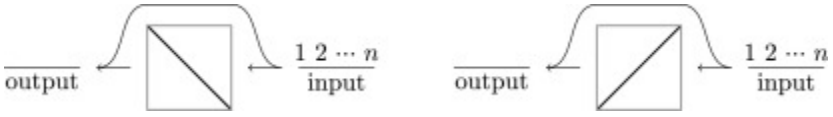


Fig. 2. The $\mathrm{Av}(12)$-machine, which generates $\mathrm{Av}(312)$, and the $\mathrm{Av}(21)$-machine, which generates $\mathrm{Av}(321)$. The internal lines in diagrams of this type represent the allowed positions of the elements stored in the machine, so in the first case any entries in the machine must be decreasing when read left to right, and in the second case they must be increasing.

Before beginning the general study of $\mathcal{C}$-machines, we consider one more specific example, the $\mathrm{Av}(21)$-machine, illustrated on the right of Fig. 2. In this machine we may only push into the rightmost end of the container, and since pops only occur from the far left of the machine, the bypass operation is necessary. We claim that this machine generates the class $\mathrm{Av}(321)$. It is evidently impossible for the machine to generate 321, as the 3 would have to be the result of a bypass while the 21 pattern lies in the container, which is not possible. In the other direction, we know that permutations in $\mathrm{Av}(321)$ consist of two increasing subsequences: their left-to-right maxima (the entries $\pi(j)$ satisfying $\pi(j) > \pi(i)$ for all $i < j$) and their non-left-to-right maxima. Upon reading the next symbol from the input, if it is to be a left-to-right maximum we first pop all entries in the container that come before it and then perform a bypass to put it in the correct position in the output, while if the next symbol from the input is not a left-to-right maximum we can simply push it into the container. When the input is empty, we finish by *flushing* (popping all the entries of) the container.

It is well-known that $\mathrm{Av}(312)$ and $\mathrm{Av}(321)$ are both counted by the Catalan numbers, so the $\mathrm{Av}(21)$- and $\mathrm{Av}(12)$-machines generate equinumerous permutation classes. This is no accident. Indeed, Section 2 shows how the description of $\mathrm{Av}(312)$ and $\mathrm{Av}(321)$ via $\mathcal{C}$-machines implicitly defines a bijection between these two classes which preserves the location and value of left-to-right maxima. This was observed in a similar context by Doyle [14].

1.3. The main property

It follows directly from the definitions that if $\mathcal{C}$ is a permutation class then so is the collection of permutations output from the C-machine. The following theorem shows that there is a close connection between the bases of these two classes.

Theorem 1.1

For any set B of permutations, the $\mathrm{Av}(B)$-machine generates the class

$$\mathrm{Av}(1 \ominus B) = \mathrm{Av}(\{1 \ominus \beta : \beta \in B\}).$$

Proof

The $\mathrm{Av}(B)$-machine cannot generate any permutation of the form $1 \ominus \beta$ for $\beta \in B$; to do so, the container would have to contain a copy of β at the point when the first entry of $1 \ominus \beta$ was next in the input.

For the converse, suppose that π avoids $1 \ominus \beta$ for all $\beta \in B$. Label the positions of the left-to-right maxima of π as $1 = i_1 < i_2 < \cdots < i_k$. At the moment that $\pi(i_j)$ is the next symbol of the input, all entries which lie before it in π are smaller than it (because it is a left-to-right maximum) so we may suppose that these entries have already exited or bypassed the container. Thus at this moment, the entries of π which lie to the right and are smaller than $\pi(i_j)$ should be in the container. This is possible because these entries avoid all of the permutations in B (because π avoids $1 \ominus B$). Thus upon reaching this point, we may bypass the container to place $\pi(i_j)$ directly in the output. We may then output all entries of the container which lie to the left

of $\pi(i_{j+1})$ in π, and proceed as before. At the end of the process, we flush the container to complete the generation of π. □

The simple characterization provided by Theorem 1.1 allows us to tell immediately if a class is generated by a $\mathcal{C}$-machine: a class is generated by a $\mathcal{C}$-machine if and only if all of its basis elements begin with their largest entries. It also allows us to tell *what* $\mathcal{C}$-machine generates a given class. In particular, let us consider a question raised by Miklós Bóna at the conference *Permutation Patterns 2007* [33, Question 4]. Atkinson, Murphy, and Ruškuc [5] showed that the permutation class sortable by two "ordered" stacks in series, despite having the infinite basis

$$\{2(2k-1)416385\cdots(2k)(2k-3): k \geq 2\},$$

is equinumerous to the class Av(1342), first counted by Bóna [8] (a simpler proof of this Wilf-equivalence result has since been given by Bloom and Vatter [7]). Bóna asked

"Is there a natural sorting machine / algorithm which can sort precisely the class $\mathrm{Av}(1342)$?"

The answer (up to symmetry) is yes: the symmetric class $\mathrm{Av}(4213)$ is generated by the $\mathrm{Av}(213)$-machine.

From the form of the basis for the class produced by a $\mathcal{C}$-machine (or from a simple consideration of its operation) all such classes are closed under the sum operation. It then follows immediately from Fekete's Lemma that all such classes have a proper growth rate (see Arratia [4]).

1.4. Operation of the machine

For any class $\mathcal{C}$, the $\mathcal{C}$-machine seemingly has three operations at its disposal: bypass, push, and pop. For enumerative purposes we must establish a *unique generation sequence* for every permutation that can be generated. To this end, we adopt conventions that handle two situations where non-uniqueness could arise:

(U1) we should pop from the container whenever possible, and
(U2) we should bypass the container whenever possible.

Another way to phrase these two rules is that in trying to generate a particular permutation π from the input sequence $12\cdots n$ if at some point the next symbol of π is the first symbol in the container then it should be output immediately (by U1), while if it is the next symbol on the input then it should be produced by an immediate bypass (by U2). The rules (U1) and (U2) correspond to choosing the "leftmost" possible action at all times. Another valuable observation is that (U1) and (U2) together imply that in any generation sequence, no pop will immediately follow a push, because otherwise the pop should have either been a bypass or occurred earlier. In our resulting functional equations, this issue will frequently arise as a flag which indicates whether pops are permitted in the corresponding state. Our next result verifies that (U1) and (U2) indeed guarantee uniqueness.

Proposition 1.2

For any class $\mathcal{C}$ and any permutation π that can be generated by the $\mathcal{C}$-machine, there is a unique generation sequence satisfying (U1) and (U2) that produces π from the $\mathcal{C}$-machine. Moreover, the left-to-right maxima of π are exactly those symbols produced by bypass operations.

Proof

Suppose that π can be generated by the $\mathcal{C}$-machine. Clearly we can find a generation sequence for π which satisfies (U1) and (U2), so it suffices to show that this generation sequence is uniquely determined by π, (U1), and (U2).

At the point when π(i) is the next symbol in the input, all smaller symbols lie either in the container or the output. By (U1), we must first pop all symbols that we can before doing anything to $\pi(i)$. By (U2), if $\pi(i)$ is a left-to-right maximum, it must bypass the container. Otherwise $\pi(i)$ is not a left-to-right maximum so it must be pushed into the container (since it is preceded in π by some greater symbol which is still in the input) and since container symbols are output in left-to-right order its placement relative to the other entries (if any) currently held in the container is uniquely determined by its position in π. □

1.5. Structure of the paper

As will be demonstrated, if a class can be generated by a $\mathcal{C}$-machine, then it is fairly automatic to use the machine to determine a set of functional equations (including catalytic variables) for its generating function. Roughly speaking the remainder of this paper consists of an exploration of that claim in a series of examples where dealing with the resulting generating functions becomes more and more difficult.

We begin with the classical cases enumerated by the Catalan and Schröder numbers in Section 2. Here the solutions of the functional equations can be derived easily using the kernel method. There is still value in bringing the $\mathcal{C}$ -machine context into play here as it establishes *uniform* arguments for these Wilf-equivalences and also provides bijections between the corresponding classes that preserve both the values and positions of the left-to-right maxima.

In Section 3 we consider two cases (the "Fibonacci machines") where algebraic solutions of the functional equations can still be rigorously obtained.

Section 4 establishes some general results which show that for very small classes $\mathcal{C}$, the corresponding classes produced by their $\mathcal{C}$-machines have easily computed enumerations.

Already in some of the earlier sections, but particularly when we proceed to Section 5, it is clear that while the translation from $\mathcal{C}$-machines to functional equations is 'fairly' automatic, it can require some effort to simplify these functional equations into a form that either permits a solution or allows for the efficient generation of a large number of terms. In the latter case it is often more useful to consider how to represent the internal state of the $\mathcal{C}$-machine using a small number of parameters. When this is possible, dynamic programming approaches allow for the generation of thousands of initial terms of the corresponding sequences.

Alas, sometimes this simplification proves impossible, as for the notoriously unenumerated class $\mathrm{Av}(4231)$. While we may view this class as the output of the $\mathrm{Av}(231)$-machine, that perspective does not improve our knowledge of its enumeration. However, the C-machine approach does allow us to compute a great number of terms for some of its subclasses. To pick an example we find particularly alluring, in Section 5 we show how to generate 5,000 terms of the enumeration of the class $\mathrm{Av}(4231,4123,4312)$. Despite the abundance of data we have for this example, we are not able to fit its generating function to any algebraic differential equation. Interestingly this means that in the chain of classes

$$\mathrm{Av}(4231,4123,4312) \subset \mathrm{Av}(4231,4312) \subset \mathrm{Av}(4231),$$

the first class is easy to enumerate (we can compute terms in polynomial time) seems to lack a D-finite generating function, the second has an algebraic generating function (see Section 2.2 where we analyze it as a $\mathcal{C}$-machine), and the third seems very difficult to enumerate (the current record is 50 terms, computed by Conway, Guttmann, and Zinn-Justin [13] building on the work of Johansson and Nakamura [21] and Conway and Guttmann [12]).

Noonan and Zeilberger [28] conjectured in 1996 that every finitely based permutation class has a D-finite generating function. Zeilberger changed his mind about the conjecture less than a decade later (see [16]) and Garrabrant and Pak [17] have recently disproved it. We believe that the class

$\mathrm{Av}(4231,4123,4312)$

represents a good candidate to be the first *concrete* counterexample to the false conjecture.

2. Catalan and Schröder classes

In this section we show how the perspective of $\mathcal{C}$-machines allows us to define length-preserving bijections between a number of classes enumerated by the Catalan and Schröder numbers. While this ground is well-trodden, we believe that at the very least the $\mathcal{C}$-machine perspective presents a particularly straightforward view of these Wilf-equivalences.

2.1. Catalan classes

The rules (U1) and (U2) of Section 1.4 implicitly give a bijection between $\mathrm{Av}(312)$ and $\mathrm{Av}(321)$. When generating a permutation with either the $\mathrm{Av}(12)$- or $\mathrm{Av}(21)$-machine, we must pop whenever possible and all left-to-right maxima must bypass the container. Moreover, since in either case the contents of the container must form a monotone sequence, whenever we push into the container, there is a unique position for the new entry to be placed. In fact, this argument establishes that there is a unique bijection between $\mathrm{Av}(312)$ and $\mathrm{Av}(321)$ that preserves the locations and values of left-to-right maxima. This bijection is equivalent, by symmetry, to one presented by Knuth [23].

Note that this bijection also restricts to a bijection between permutations that can be generated by the $\mathrm{Av}(12, k \cdots 21)$ - and $\mathrm{Av}(21,12 \cdots k)$ -machines, implying that the classes $\mathrm{Av}(312, (k+1) \ldots 21)$ and $\mathrm{Av}(321, (k+1)12 \ldots k)$ are Wilf-equivalent. This result was first established by Chow and West [11], who showed that the generating functions of these classes are quotients of Chebyshev polynomials. Of course, these generating functions simply count Dyck paths of maximum height k.

We now consider a different viewpoint which will become necessary when we analyze more complicated machines. We can think of the $\mathrm{Av}(21)$-machine operating under the rules (U1) and (U2) as being in one of two states that we call “can pop” and “can't pop”. The machine is in the “can't pop” state whenever we have just pushed a symbol into the container and in the “can pop” state at all other times, as shown in Fig. 3.

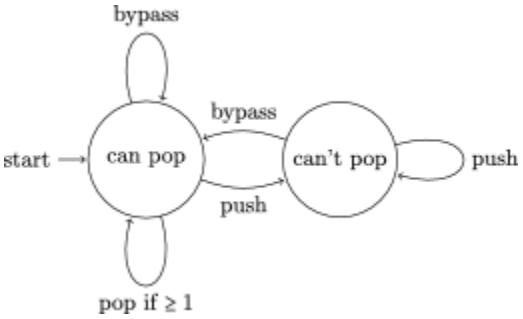


Fig. 3. An automaton representing the $\mathrm{Av}(21)$-machine.

Let $f(x,u)$ denote the generating function for paths to the “can pop” state, where x tracks the number of output symbols, and u tracks the number of symbols in the container. Also let $g(x,u)$ denote the generating function for paths to the “can't pop” state with the same variables. By considering all possible transitions among these two states, we derive the system of equations

$$\begin{aligned} f(x,u) &= 1 + x(f(x,u) + g(x,u)) + \frac{x}{u}(f(x,u) - f(x,0)), \\ g(x,u) &= u(f(x,u) + g(x,u)). \end{aligned}$$

Then by a simple application of the kernel method[2] we get

$$f(x,0) = (1-\sqrt{1-4x})/2x.$$

2.2. The Schröder classes

It is an easy computation to show that the classes defined by avoiding two patterns of length four (the *2×4 classes*) form 56 symmetry classes. After a significant amount of work [9], [24], [25], [26], [27], it has been shown that these 56 symmetry classes fall into 38 Wilf equivalence classes, of which explicit generating functions have been found for all but 5. By far the largest of these Wilf equivalence classes consists of 10 symmetry classes enumerated by the Schröder numbers (this Wilf equivalence class was found by Kremer [24], [25]). Of these 10 symmetry classes, 6 can be generated by $\mathcal{C}$-machines, in a completely parallel manner, as we describe in this section.

The first Schröder class we consider is $\mathrm{Av}(4312,4213)$, which is generated by the $\mathrm{Av}(312,213)$-machine shown in Fig. 4. As indicated by the dark lines in this figure, permutations in $\mathrm{Av}(312,213)$ consist of an increasing sequence followed by a decreasing sequence.

Fig. 4. The $\mathrm{Av}(312,213)$-machine generates a Schröder class.

By (U1) and (U2), pops and bypasses in the $\mathrm{Av}(312,213)$-machine function the same as they do in the $\mathrm{Av}(21)$-machine, but pushes function differently. When the container is empty there is only one position to push into, and when the container is nonempty there are two positions to push into: either immediately to the left of the maximum entry in the container or immediately to the right of this entry. By making a small variation to the functional equations for the $\mathrm{Av}(21)$-machine, we are led to the system

$$\begin{aligned} f(x,u) &= 1 + x(f(x,u) + g(x,u)) + \frac{x}{u}(f(x,u) - f(x,0)), \\ g(x,u) &= 2u((f(x,u) - f(x,0)) + g(x,u)) + uf(x,0). \end{aligned}$$

Here the $f(x,u)$ equation has stayed the same, but the $g(x,u)$ equation has changed to reflect the number of positions we may push into.

This example is sufficiently simple to solve by hand using the kernel method and yields

$$f(x,0) = \frac{3 - x - \sqrt{1-6x+x^2}}{2}.$$

Proposition 2.1

The permutation classes $\mathrm{Av}(4312,4213)$, $\mathrm{Av}(4132,4231)$, $\mathrm{Av}(4312,4231)$, $\mathrm{Av}(4213,4132)$, $\mathrm{Av}(4321,4312)$, *and* $\mathrm{Av}(4213,4123)$ *are all enumerated by the Schröder numbers. Furthermore, there are length-preserving bijections between them that also preserve the location and value of the left-to-right maxima.*

Proof

Fig. 4 and Fig. 5 show the six classes whose associated machines generate the Schröder classes listed. In each case it is evident that, when the machine contains one or more elements there are exactly two ways to insert a new maximum element. Recall that Proposition 1.2 guarantees that given any permutation π that can be generated by a $\mathcal{C}$-machine, there is a unique generation sequence satisfying (U1) and (U2). Since all these

machines have the same generation sequences this provides length preserving bijections between the corresponding classes produced, and these bijections preserve the location and values of left-to-right maxima because these are precisely the elements of the output produced by bypass operations. □

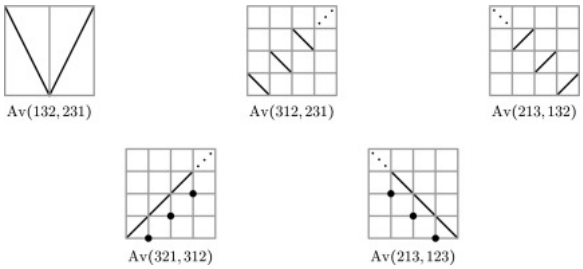


Fig. 5. Five classes whose machines generate Schröder classes.

As far as we are aware, bijections between these classes preserving the location and values of the left-to-right maxima have not been presented in the literature before, although their existence also follows from the results of Bloom and Elizalde [6, Section 6].

The remaining four Schröder classes are:

$$\mathrm{Av}(4312,3412), \mathrm{Av}(4213,2413), \mathrm{Av}(4213,3214), \mathrm{Av}(3142,2413).$$

None of these (or any of their symmetries) have both basis elements beginning with a 4 and therefore they cannot be enumerated by the mechanisms of $\mathcal{C}$-machines as we are presenting them here. While it might be possible to generalize or vary the operation of $\mathcal{C}$-machines to account for these classes, it is easy to see by direct computation that there cannot be bijections that preserve the positions and values of left-to-right maxima between the six Schröder classes we have considered and the other four Schröder classes. Thus such variations would at the very least modify the type of permutation statistics that are preserved by a consideration of distinct $\mathcal{C}$-machines operating with a common operation sequence.

3. Fibonacci machines

Here we consider the class of permutations $\mathcal{F}_{\oplus}$ formed by sums of the permutations 1 and 21 and the symmetric class of permutations $\mathcal{F}_{\ominus}$ formed by skew sums of the permutations 1 and 12 as shown in Fig. 6 (the presence of two dots in each cell indicates that we may put zero, one, or two entries in each cell). We call these classes the *Fibonacci* classes, as the number of permutations of length n in each class is the nth Fibonacci number, F_n, with initial conditions $F_0 = F_1 = 1$.

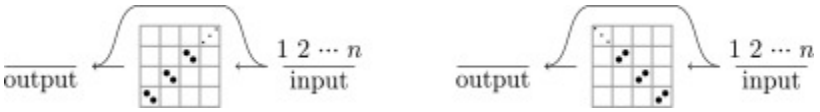


Fig. 6. The $\mathcal{F}_{\oplus}$ - and $\mathcal{F}_{\ominus}$ -machines.

At first glance it might seem that if two permutation classes $\mathcal{C}$ and $\mathcal{D}$ are Wilf-equivalent then so should be the classes generated by the $\mathcal{C}$-machine and the $\mathcal{D}$ -machine. However, the operation of a machine is not symmetric: input arrives at the top, but output is produced from the left. Unless there is a bijection underlying the original Wilf-equivalence that respects this asymmetry, the corresponding machines will behave differently. To give a concrete example of why this is the case with respect to the $\mathcal{F}_{\oplus}$ - and $\mathcal{F}_{\ominus}$ -machines, suppose we fill the $\mathcal{F}_{\oplus}$ -machine with 2143, then perform a bypass, and then a pop. The container will then hold 143, and there is a unique way to perform a push, then a bypass, and then empty the machine. On the other hand, the

analogous generation sequence applied to the $\mathcal{F}_{\ominus}$ -machine would tell us to fill it with 3412, then perform a bypass and a pop. At that point the container will hold 412, leaving us with two ways to perform a push (resulting in either 5412 or 4512), then a bypass, and then to empty the machine.

It is known that $\mathcal{F}_{\oplus} = \mathrm{Av}(231,312,321)$ and $\mathcal{F}_{\ominus} = \mathrm{Av}(123,132,213)$. Thus Theorem 1.1 implies that the $\mathcal{F}_{\oplus}$ -machine generates the class $\mathrm{Av}(4231,4312,4321)$, while the $\mathcal{F}_{\ominus}$ -machine generates the class $\mathrm{Av}(4123,4132,4213)$. Note that these classes are both subclasses of Schröder classes considered in the previous section.

3.1. The $\mathcal{F}_{\oplus}$ -machine

In this subsection we consider the class $\mathrm{Av}(4231,4312,4321)$ generated by the $\mathcal{F}_{\oplus}$ -machine.

Theorem 3.1

The permutation class $\mathrm{Av}(4231,4312,4321)$ *generated by the* $\mathcal{F}_{\oplus}$ *-machine has a generating function which is algebraic of degree 4 and growth rate*

$$\frac{(3-\sqrt{5})(7+3\sqrt{5}+2\sqrt{22+10\sqrt{5}})}{4},$$

approximately 5.16207.

Proof

We start by crafting an automaton to represent the $\mathcal{F}_{\oplus}$ -machine, similar to that in Fig. 3 for the $\mathrm{Av}(21)$-machine. However, this automaton is more complicated than any of the corresponding automata for the Catalan and Schröder classes due to one important fact: the number of places where we can push the next element into the machine can vary between 1 and 2 (in the machines for the Catalan classes it was always 1, and in the machines for the Schröder classes, it was always 2 so long as the machine was non-empty). As such, the automaton for the $\mathcal{F}_{\oplus}$ -machine, shown in Fig. 7, has 5 states: E represents an empty machine, S_p (resp. S_n) represent states in which the rightmost layer has only one entry and pops are permitted (resp. forbidden) and D_p (resp. D_n) represent states in which the rightmost layer has two entries and pops are permitted (resp. forbidden).

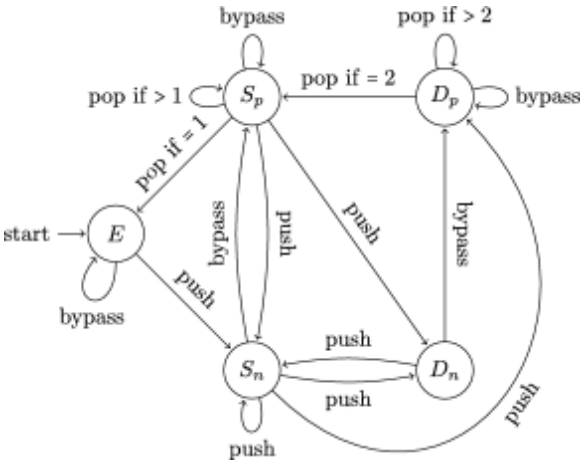


Fig. 7. An automaton representing the $\mathcal{F}_{\oplus}$ -machine.

Let $E(x), S_p(x,u), S_n(x,u), D_p(x,u), D_n(x,u)$ be the generating functions that track states as described above, such that x counts the number of entries that have been output (via pops and bypasses) and u counts the

number of entries in the machine *excluding the rightmost layer*. The automaton in Fig. 7 translates to the following system of functional equations.

$$\begin{aligned}
E(x) &= 1 + xE(x) + xS_p(x,0) \\
S_p(x,u) &= x(S_n(x,u) + S_p(x,u)) + \frac{x}{u}(S_p(x,u) - S_p(x,0)) + x(D_p(x,0)) \\
S_n(x,u) &= E(x) + u(S_n(x,u) + S_p(x,u)) + u^2(D_n(x,u) + D_p(x,u)) \\
D_p(x,u) &= x(D_n(x,u) + D_p(x,u)) + \frac{x}{u}(D_p(x,u) - D_p(x,0)) \\
D_n(x,u) &= S_n(x,u) + S_p(x,u)
\end{aligned}$$

Note that because, for example, in $S_p(x,u)$ the variable u tracks the contents of the machine not considering the rightmost layer, the generating function $S_p(x,0)$ represents states with only a single entry in the machine.

We can use the first, third, and fifth equation to eliminate $E(x)$, $S_n(x,u)$, and $D_n(x,u)$ from the system, leaving

$$\begin{aligned}
S_p(x,u) &= \frac{x(1-u^2)}{u(1-u-u^2)}S_p(x,u) - \frac{1-x-u-u^2+xu^2}{u(1-x)(1-u-u^2)}S_p(x,0) \\
&\quad + \frac{xu^2}{1-u-u^2}D_p(x,u) + xD_p(x,0) + \frac{x}{(1-x)(1-u-u^2)} \\
D_p(x,u) &= \frac{x}{1-u-u^2}S_p(x,u) + \frac{x^2}{(1-x)(1-u-u^2)}S_p(x,0) \\
&\quad + \frac{x(1-2u^2)}{u(1-u-u^2)}D_p(x,u) - \frac{x}{u}D_p(x,0) + \frac{x}{(1-x)(1-u-u^2)}
\end{aligned}$$

Solving the second equation for $D_p(x,u)$ and substituting into the first leaves a single equation in terms of $S_p(x,u)$, $S_p(x,0)$, and $D_p(x,0)$. Solving that equation for $S_p(x,u)$ and clearing denominators gives

(†) $K(x,u)S_p(x,u) = R_1(x,u)S_p(x,0) + R_2(x,u)D_p(x,0) + R_3(x,u)$

With

$$\begin{aligned}
K(x,u) &= (x-1)(u^4 - 3u^3x + u^2x^2 + u^3 - ux^2 - u^2 + 2xu - x^2) \\
R_1(x,u) &= -x(u^3x - u^2x^2 - u^3 + 2u^2x + ux^2 - u^2 - xu + x^2 + u - x) \\
R_2(x,u) &= xu(x-1)(u^3 - u^2x + u^2 - u + x) \\
R_3(x,u) &= -xu(xu - u + x).
\end{aligned}$$

Theorem 2 of Bousquet-Mélou and Jehanne [10] confirms that there are four fractional power series $u_1(x), \ldots, u_4(x)$, counted with multiplicity, such that $K(x,u_i(x)) = 0$. By checking initial terms, we find in fact that the $u_i(x)$ are distinct. Moreover, substituting any $u_i(x)$ into Equation (†) produces a new equation

$$0 = R_1(x,u_i(x))S_p(x,0) + R_2(x,u_i(x))D_p(x,0) + R_3(x,u_i(x)).$$

Combining the equation above for $i = 1,2$ with the two equations $K(x,u_1(x)) = 0$ and $K(x,u_2(x)) = 0$ gives a system of four polynomial equations from which we might hope to eliminate the variables u_1, u_2, and $D_p(x,0)$. This elimination can be performed efficiently through the use of Gröbner bases.[3] As a result, we find that

$$0 = x \cdot (1 + S_p(x,0)) \cdot A(x, S_p(x,0)),$$

where

$$\begin{aligned} A(x, S_p(x,0)) &= (2x^3+8x^2-x)S_p(x,0)^4 - (x^4+3x^3-58x^2+19x-1)S_p(x,0)^3 \\ &\quad +(3x^4-30x^3+130x^2-56x+7)S_p(x,0)^2 \\ &\quad -(x^4+3x^3-58x^2+19x-1)S_p(x,0) \\ &\quad +(2x^3+8x^2-x). \end{aligned}$$

Hence, $A(x, S_p(x,0))$ is the minimal polynomial for the algebraic generating function $S_p(x,0)$. One can combine this minimal polynomial with the equation $E(x) = 1 + xE(x) + xS_p(x,0)$ (for example, with a resultant calculation) to find the minimal polynomial for $E(x)$:

$$\begin{aligned} (2x^2+8x-1)E(x)^4 + (x^3+4x^2-46x+5)E(x)^3 \\ +(3x^3-21x^2+94x-9)E(x)^2 \\ +(x^3+12x^2-82x+7)E(x) \\ +3x^2+26x-2 \quad = 0. \end{aligned}$$

At this point Maple can explicitly solve for $E(x)$ and inspection reveals that the growth rate of the class is as stated in the theorem. □

The enumeration of this class is given by sequence A257561 in the OEIS [31].

3.2. The $\mathcal{F}_{\ominus}$ -machine

The $\mathcal{F}_{\ominus}$ -machine differs from the $\mathcal{F}_{\oplus}$ -machine because in the $\mathcal{F}_{\ominus}$ -machine a pop can reduce the size of the leftmost layer—which in this case is the layer we might push into—thereby opening up more possibilities for the next push and forcing us in some sense to remember the size of the layer to its right (in case a pop empties the leftmost layer).

Theorem 3.2

The class $\mathrm{Av}(4123,4132,4213)$ *generated by the* $\mathcal{F}_{\ominus}$ *-machine has generating function* $s(x)$ *which is algebraic of degree 3 and its growth rate is*

$$\frac{67240 + (779\sqrt{57} - 1927)(1502 + 342\sqrt{57})^{1/3} - (19\sqrt{57} - 457)(1502 + 342\sqrt{57})^{2/3}}{40344},$$

approximately 5.21914*.*

Proof

Because of the difference between the $\mathcal{F}_{\ominus}$ - and $\mathcal{F}_{\oplus}$ -machines noted above, a direct approach to computing the generating function of $\mathrm{Av}(4123,4132,4213)$ based around an iterative scheme for accounting for the generation sequences is not feasible. For this reason we construct a context-free grammar instead.

Let W_n be the language of words (tracking states of the $\mathcal{F}_{\ominus}$ -machine) that begin from a state where the leftmost layer is a single entry with no immediate legal pop and end with the same entry alone in the leftmost layer with a pop now allowed. Similarly, R_n will be the language of words beginning in a state where the leftmost layer contains two entries with no immediate legal pop and ending with the same two entries in the leftmost layer with a pop now allowed. Let W_p (resp., R_p) be the language of words that start with a single entry (resp., two entries) in the leftmost layer with a legal pop allowed and end with the same single entry (resp., two entries) in the leftmost layer with a legal pop allowed.

These definitions yield the following context-free grammar for legal operation sequences in the $\mathcal{F}_{\ominus}$ -machine.

$$
\begin{array}{lcl|l|l|l|l}
S & \longrightarrow & \epsilon & | & xS & | & (+w)W_n(-w)S \\
W_p & \longrightarrow & \epsilon & | & xW_p & | & (+w)W_n(-w)W_p & | & (+r)R_n(-r)W_p \\
W_n & \longrightarrow & & & xW_p & | & (+w)W_n(-w)W_p & | & (+r)R_n(-r)W_p \\
R_p & \longrightarrow & \epsilon & | & xR_p & | & (+w)W_n(-w)R_p \\
R_n & \longrightarrow & & & xR_p & | & (+w)W_n(-w)R_p
\end{array}
$$

The nonterminals S, W_p, and R_p each have a production rule to ϵ because the starting condition satisfies the ending condition for each of these languages, whereas this is not true for W_n and R_n. The production rules of the form $T \longrightarrow xT$ represent a bypass operation. As popping is always permitted after a bypass, the bypass is always followed by a state in which popping is legal (e.g., $W_n \longrightarrow xW_p$).

The remaining production rules correspond to pushing an element in a new layer (represented by $(+w)$) or adding an entry to an existing layer of size one (represented by $(+r)$), then performing an appropriate sequence W_n or R_n, then popping the entry added earlier (represented by $(-w)$ or $(-r)$). Lastly, each of these production rules ends by allowing a repeated occurrence of either W_p in the case where the production symbol is W_p or W_n or of R_p in the case where the production symbol is R_p or R_n.

This context-free grammar is unambiguous because in every rule the start symbols of the various cases are distinct. Hence, we can translate the grammar to equations, replacing $(-w)$ and $(-r)$ with x to keep track of pop operations, and ignoring $(+w)$ and $(+r)$ because we do not need to keep track of pushes. This yields the following system.

$$
\begin{aligned}
s &= 1 + xs + xw_n s \\
w_p &= 1 + xw_p + xw_n w_p + xr_n w_p \\
w_n &= xw_p + xw_n w_p + xr_n w_p \\
r_p &= 1 + xr_p + xw_n r_p \\
r_n &= xr_p + xw_n r_p
\end{aligned}
$$

Another Gröbner basis calculation reveals that s satisfies

$$1 + (x-1)s(x) - xs(x)^2 + xs(x)^3.$$

This implies that the growth rate of the class is as stated in the theorem. □

The enumeration of the class is given by sequence A106228 in the OEIS [31].

4. Machines for restricted classes

As the reader may have noticed, the analysis of $\mathcal{C}$-machines typically depends on the specific structure of the class $\mathcal{C}$ itself. However, by placing restrictions on the characteristics of the class $\mathcal{C}$ it is possible to establish some general characteristics of the classes corresponding to the associated $\mathcal{C}$-machine. In this section we explore three such restrictions for which we are able to establish general results: to finite, bounded, and polynomial classes.

4.1. Finite classes

First we consider the case where $\mathcal{C}$ is a finite class. Following Albert, Atkinson, and Ruškuc [2], we say that the *rank* of the entry π(i) is the number of entries below it and to its right. When $\mathcal{C}$ is finite, the class of permutations that can be generated by the $\mathcal{C}$-machine necessarily has bounded rank. Moreover, because every finite class has a finite basis (an easy consequence of the Erdős–Szekeres Theorem), the class of permutations that can be generated by the $\mathcal{C}$-machine has a finite basis, and the results of [2] imply that this class has a rational generating function.[4]

Theorem 4.1

If $\mathcal{C}$ is a finite class then the class of permutations that can be generated by the $\mathcal{C}$-machine has a rational generating function.

More in the spirit of the "$\mathcal{C}$-machine approach" an alternative way to prove this theorem would be to note that for a finite class $\mathcal{C}$ there are only finitely many states that the $\mathcal{C}$-machine can take (at most twice the total number of permutations in $\mathcal{C}$ allowing for the "can't pop", "can pop" distinction). Therefore, after choosing an alphabet that allows us to represent all the different ways that a new maximum element can be added to the machine, the $\mathcal{C}$-machine can be thought of as a finite state automaton. The conclusion of the theorem follows immediately.

4.2. Polynomial classes

We next consider the case where $|\mathcal{C}_n|$ is bounded by a polynomial (in n), in which case we call $\mathcal{C}$ a *polynomial class*. Kaiser and Klazar [22] established two significant results regarding polynomial classes. First, they showed that polynomial classes are actually enumerated by polynomials for sufficiently large n (i.e., they are not just polynomially bounded). Second, they showed that if the enumeration of a class is ever less than the *n*th combinatorial *Fibonacci* number (defined by $F_0 = F_1 = 1$ and $F_n = F_{n-1} + F_{n-2}$) then the class is a polynomial class. This second statement is referred to as the Fibonacci Dichotomy. Later, Huczynska and Vatter [20] reproved the Fibonacci Dichotomy using what are known as grid classes and gave an explicit characterization of polynomial classes. While we do not need to appeal to the details of this characterization, we do require the following fact that follows from it.

Proposition 4.2

Every polynomial class is finitely based.

Proposition 4.2 is explicitly proved in the conclusion of Huczynska and Vatter [20] and also follows from the later and more general Vatter [34, Theorem 6.2].

Our result about polynomial classes requires one further notion. Inspired by Wilf's influential *Monthly* article "What is an answer?" [36], Zeilberger [37] defined a *Wilfian formula* for the sequence {an} to be a polynomial-time (in n) algorithm that computes an. For example, an algebraic generating function can easily be converted into a Wilfian formula (one needs only to compute derivatives), but many sequences that do not have algebraic generating functions still have Wilfian formulas (e.g., the Catalan numbers modulo 2).

Theorem 4.3

If $\mathcal{C}$ is a polynomial class then the class of permutations that can be generated by the $\mathcal{C}$-machine has a Wilfian formula.

Proof

Let $\mathcal{C}$ be a polynomial class and choose a polynomial $c(n)$ such that $|\mathcal{C}_n| \leq c(n)$ for all n. By Proposition 4.2, $\mathcal{C} = \mathrm{Av}(B)$ for a finite set *B*. Let m denote the length of the longest basis element of *B*. Thus we can determine whether a permutation of length n lies in $\mathcal{C}$ in time $b(n) = |B|\binom{n}{m}$. We seek to show that there is a polynomial $p(n)$ such that we can determine the number of permutations of length *n* that can be generated by the $\mathcal{C}$-machine in time at most $p(n)$.

To accomplish this, we create an automaton that has two states for each permutation of length at most *n* in $\mathcal{C}$. Of these two states, one corresponds to the "can pop" condition and the other to the "can't pop" condition, while the permutation associated to the state records the order isomorphism type of the contents of the machine. We can build this automaton by working up from the states corresponding to the empty permutation

by considering all possible pushes, pops (if the "can pop" condition is true for that state), and bypasses. For each state whose corresponding permutation has length k, there are at most $k+3$ such actions. Pops and bypasses are trivial to analyze, while for each possible push we can determine if the push leads to a permutation in $\mathcal{C}$ in time $b(k+1)$. Therefore we can construct this automaton in time at most

$$\sum_{k=0}^{n-1}(k+3)b(k+1)c(k),$$

which is a polynomial of degree at most $2+m+\deg c$. To compute $|\mathcal{C}_n|$ from this automaton we simply count the number of closed walks beginning and ending at the state corresponding to the empty permutation that contain a total of n pops and bypasses. As the automaton has only a polynomial number of states, the number of these walks can be computed in polynomial time. □

The argument above carries through almost directly when $\mathcal{C}$ is not quite a polynomial class, but instead the sum closure of a polynomial class. For example the permutations in $\mathrm{Av}(231,321)$ are all sums of permutations of the form $k12\cdots(k-1)$. The corresponding machine is analyzed in the next section.

Needless to say, the algorithm described in the proof of Theorem 4.3 should not be implemented. To obtain a more practical algorithm for enumerating these $\mathcal{C}$-machines, one would want to implement a dynamic programming algorithm exploiting the specific structure of $\mathcal{C}$. We present several examples of this in Section 5.

4.3. Bounded classes

We conclude this section with the consideration of *bounded classes*: those classes $\mathcal{C}$ for which there exists an integer c such that $|\mathcal{C}_n| \le c$ for all $n \ge 0$. Obviously bounded classes are a special case of polynomial classes, but because our result is stronger we must describe the structure of bounded classes in more detail. In doing so we follow Homberger and Vatter [19].

An *interval* in a permutation is a sequence of contiguous entries whose values form an interval of natural numbers. A *monotone interval* is an interval in which the entries are monotone (increasing or decreasing). Given a permutation σ of length m and nonempty permutations $\alpha_1,\ldots,\alpha_m$, the *inflation* of σ by $\alpha_1,\ldots,\alpha_m$ is the permutation $\pi=\sigma[\alpha_1,\ldots,\alpha_m]$ obtained by replacing each entry $\sigma(i)$ by an interval that is order isomorphic to α_i, while maintaining the relative order of the intervals themselves. For example,

$$3142[1,321,1,12]=6\,321\,7\,45.$$

We define a *peg permutation* to be a permutation where each entry is decorated with a +, −, or •, such as

$$\tilde{\rho}=3^{\bullet}1^{-}4^{\bullet}2^{+}$$

The *grid class* of the peg permutation $\tilde{\rho}$, denoted $\mathrm{Grid}(\tilde{\rho})$, is the set of all permutations that may be obtained by inflating ρ (the underlying, non-decorated version of $\tilde{\rho}$) by monotone intervals of type determined by the signs of $\tilde{\rho}$: $\rho(i)$ may be inflated by an increasing (resp., decreasing) interval if $\tilde{\rho}(i)$ is decorated with a + (resp., −) while it may only be inflated by a single entry (or the empty permutation) if $\tilde{\rho}(i)$ is dotted. Thus if $\pi\in\mathrm{Grid}(\tilde{\rho})$ then its entries can be partitioned into monotone intervals which are "compatible" with $\tilde{\rho}$.

Given a set $\tilde{G}$ of peg permutations, we denote the union of their corresponding grid classes by

$$\mathrm{Grid}(\tilde{G}) = \bigcup_{\tilde{\rho} \in \tilde{G}} \mathrm{Grid}(\tilde{\rho}).$$

In their characterization of polynomial classes, Huczynska and Vatter [20] proved that every polynomial class is contained in $\mathrm{Grid}(\tilde{\rho})$ for a single peg permutation $\tilde{\rho}$. From this and the work of Albert, Atkinson, Bouvel, Ruškuc, and Vatter on atomic geometric grid classes [1, Theorem 10.3], the following result follows.

Theorem 4.4

For every polynomial class $\mathcal{C}$ there is a finite set $\tilde{G}$ of peg permutations such that $\mathcal{C} = \mathrm{Grid}(\tilde{G})$.

The containment relation on $\mathbb{N}^m$ is a partial order. Thus we may define downsets (sets closed downward under containment) and upsets of vectors. The intersection of a downset and an upset is referred to as a *convex set*.

We say that $\vec{v}$ *fills* the peg permutation $\tilde{\rho}$ if $\vec{v}(i) = 1$ whenever $\tilde{\rho}(i)$ is decorated with a • and $\vec{v}(i) \geq 2$ whenever $\tilde{\rho}(i)$ is decorated with a + or −. Given any peg permutation $\tilde{\rho}$ of length m and a set of vectors $\mathcal{V} \subseteq \mathbb{P}^m$ that fill $\tilde{\rho}$, we define

$$\tilde{\rho}[\mathcal{V}] = \bigcup_{\vec{v} \in \mathcal{V}} \tilde{\rho}[\vec{v}].$$

We now have all the terminology and notation to state the relevant structure theorem.

Theorem 4.5

Homberger and Vatter [19]

For every polynomial permutation class $\mathcal{C}$ there is a finite set $\tilde{G}$ of peg permutations, each associated with its own convex set $\mathcal{V}_{\tilde{\sigma}}$ of vectors of positive integers which fill it, such that $\mathcal{C}$ can be written as the disjoint union

$$\mathcal{C} = \biguplus_{\tilde{\rho} \in \tilde{G}} \tilde{\rho}[\mathcal{V}_{\tilde{\rho}}].$$

This theorem is more useful than Theorem 4.4 for the description of an enumerative scheme leading to Theorem 4.6 below precisely because the union it describes is disjoint.

We establish our result about bounded classes using *counter automata*, which are finite state automata with the additional ability to store a single nonnegative integer called a *counter*. When determining which transition to take, a counter automaton is allowed to check if the value of the counter is 0, and during each transition the value of the counter may be incremented or decremented by 1. Equivalently, for any fixed positive integer N and all n satisfying $0 \leq n \leq N$, a counter automaton is allowed to check if the value of the counter is equal to n and is allowed to increase or decrease the counter by n. Deterministic counter automata are a proper subset of deterministic pushdown automata and therefore the languages they accept have algebraic generating functions (see Droste, Kuich, and Vogler [15, Chapter 7]).

Theorem 4.6

If $\mathcal{C}$ is a bounded class then the class of permutations that can be generated by the $\mathcal{C}$-machine has an algebraic generating function.

Proof

Suppose $\mathcal{C}$ is a bounded class and let $\tilde{G}$ and the convex sets $\mathcal{V}_{\tilde{\rho}}$ for each $\tilde{\rho} \in \tilde{G}$ be as in the statement of Theorem 4.5. We build a counter automaton whose states represent the subpermutation in the container of the $\mathcal{C}$ -machine at any point in time. However, as $\mathcal{C}$ contains infinitely many permutations (otherwise it would fall under the purview of Theorem 4.1) and a standard counter automaton must have a finite number of states, some compression is necessary.

Each $\tilde{\rho}$ comes equipped with a convex set $\mathcal{V}_{\tilde{\rho}}$ of vectors in $\mathbb{N}^{|\tilde{\rho}|}$. Only one component of these vectors is allowed to grow unboundedly as otherwise the class $\mathcal{C}$ would not be bounded. For each $\tilde{\rho} \in \tilde{G}$ let $M_{\tilde{\rho}}$ denote the maximum value of all *other* components for $\vec{v} \in \mathcal{V}_{\tilde{\rho}}$ and define

$$M = \max\left(\{M_{\tilde{\rho}} : \tilde{\rho} \in \tilde{G}\}\right).$$

That is, M is the maximum of all second-largest components over all $\vec{v} \in \mathcal{V}_{\tilde{\rho}}$ and $\tilde{\rho} \in \tilde{G}$.

Any state of the $\mathcal{C}$ -machine in which the container holds a subpermutation $\tilde{\rho}[\vec{v}]$ with $\vec{v}(i) \leq M$ for all i is simply represented by a state of the counter automaton labeled $\tilde{\rho}[\vec{v}]$. Any state of the $\mathcal{C}$-machine in which the container holds a subpermutation $\tilde{\rho}[\vec{v}]$ with some $\vec{v}(i) > M$ is represented by a state of the counter automaton labeled

$$\tilde{\rho}[\vec{v}(1), \ldots, \vec{v}(i-1), *, \vec{v}(i+1), \ldots, \vec{v}(|\tilde{\rho}|)].$$

Here the $*$ symbol represents an inflation of size at least $M + 1$, and it is this parameter that the counter keeps track of by storing the value $\min\{0, \vec{v}(i) - M\}$.

Next we split each state described above into two copies: one labeled “can pop” and one labeled “can't pop”. We add to this a state labeled ϵ to account for the empty machine which is both the start state and the unique accepting state. The transitions between each pair of states are readily computed by examining the allowed pushes, pops, and bypasses. Transitions to states with no ‘$*$’ marker must set the counter at 0, while transitions to states with a ‘$*$’ marker may or may not change the counter (they can also, of course, change the underlying $\tilde{\rho}$).

The counter automaton constructed above accepts all valid push/pop sequences that leave the container of the $\mathcal{C}$-machine empty. If transitions are weighted so that those corresponding to bypasses and pops have weight *x* and those corresponding to pushes have weight 1, then the weighted generating function counting accepting paths of length *n* is equal to the generating function for the class generated by the $\mathcal{C}$-machine. □

As with all the results of this section, note that Theorem 4.6 represents only a sufficient condition for algebraicity. In particular, it does not apply to any of the Schröder machines which nevertheless generate classes with algebraic generating functions.

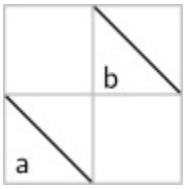
5. Potentially non-D-finite classes

Here we present four permutation classes for which, despite the fact that they can be generated by fairly simple $\mathcal{C}$-machines, we do not know (and cannot even conjecture) their generating functions. Indeed, while we

can implement the dynamic programming approach hinted at in the proof of Theorem 4.3 to obtain many terms in the counting sequence of these classes (5,000 in the first case we present), we cannot fit a D-finite generating function to any of them. The first case we present has three basis elements of length four while the three following it are so-far-unenumerated "2×4 classes". We present the first example in detail but provide only sketches of the (very similar) arguments for the remaining examples.

5.1. $\mathbf{Av(4123, 4231, 4312)}$

Theorem 1.1 implies that the class $\mathrm{Av}(4123,4231,4312)$ is generated by the $\mathrm{Av}(123,231,312)$-machine. The members of $\mathrm{Av}(123,231,312)$ can be drawn as shown below, where the labeling of entries with an a or b is for the subsequent analysis.



When the container is empty we may only push an a entry. When it contains a decreasing permutation (all of whose entries are viewed as a entries), we may push either an a or a b entry. After pushing a b entry we may only push b entries until we have popped all of the a entries, at which point all current b entries become a entries.

We represent the states of the $\mathrm{Av}(123,231,312)$-machine by triples (a, b, P) where a and b are the number of a and b entries respectively, and P is either $T(rue)$ or $F(alse)$, depending on whether popping is allowed. The preceding discussion shows:

Proposition 5.1

The transition rules for the $\mathrm{Av}(123,231,312)$*-machine are:*

$(0,0,T) \to \{(1,0,\mathrm{F}), 0,0,\mathrm{T})\},$
$(\mathrm{a}, 0, \mathrm{F}) \to \{(\mathrm{a}+1,0,\mathrm{F}), (\mathrm{a}, 1, \mathrm{F}), (\mathrm{a}, 0, \mathrm{T})\}$
$(\mathrm{a}, 0, \mathrm{T}) \to \{(\mathrm{a}+1,0,\mathrm{F}), (\mathrm{a}, 1, \mathrm{F}), (\mathrm{a}, 0, \mathrm{T}), (\mathrm{a}-1,0,\mathrm{T})\},$
$(\mathrm{a}, \mathrm{b}, \mathrm{F}) \to \{(\mathrm{a}, \mathrm{b}+1, \mathrm{F}), (\mathrm{a}, \mathrm{b}, \mathrm{T})\},$
$(\mathrm{a}, \mathrm{b}, \mathrm{T}) \to \{(\mathrm{a}, \mathrm{b}+1, \mathrm{F}), (\mathrm{a}, \mathrm{b}, \mathrm{T}), (\mathrm{a}-1, \mathrm{b}, \mathrm{T})\}(\text{for a} \geq 2),$
$(1, \mathrm{b}, \mathrm{T}) \to \{(1, \mathrm{b}+1, \mathrm{F}), (1, \mathrm{b}, \mathrm{T}), (\mathrm{b}, 0, \mathrm{T})\},$

where $a, b \geq 1$ *unless stated otherwise.*

These transition rules can be adapted to a dynamic programming algorithm, which can be used to compute the first 5,000 terms of the enumeration in a moderate amount of time. The enumeration of this class is sequence A257562 in the OEIS [31].

One can also derive a functional equation for the generating function of this class using these transition rules. Define an *A* state to be one in which there are no b entries and a *B* state to be one in which there are b entries (and therefore, also a entries). We require that popping is always permitted at the beginning of a *B* state (we explain this in more detail below). The empty state is considered an *A* state, and *A* is also the start state.

Let $A(a, x)$ be the generating function in which the coefficient of $a^k x^n$ counts the number of ways to reach an *A* state with k entries labeled a and n entries output so far. Let $B(a, b, x)$ be the generating function in which the coefficient of $a^k b^\ell x^n$ counts the number of ways to reach a *B* state with $k-1$ entries labeled a, ℓ entries

labeled b, and n entries output so far. As *B* tracks one fewer than the number of a entries, it follows that $B(0,b,x)$ enumerates the *B* states with exactly one a entry.

Proposition 5.2

The generating functions $A(a,x)$ and $B(a,b,x)$ that describe the operation of the $\mathrm{Av}(123,231,312)$*-machine satisfy the functional equations*

$$\begin{aligned} A(a,x) &= 1+\frac{x}{a}(A(a,x)-A(0,x))+aA(a,x)+xB(0,a,x), \\ B(a,b,x) &= \frac{bx}{a(1-b)(1-x)}(A(a,x)-A(0,x))+\frac{bx}{(1-b)(1-x)}B(a,b,x) \\ &\quad +\frac{x}{a(1-x)}(B(a,b,x)-B(0,b,x)). \end{aligned}$$

Proof

The *a* in the denominator in the first term in $B(a,b,x)$ accounts for the fact that $B(a,b,x)$ tracks one fewer than the number of $A(a,x)$.

An *A* state is reached from another *A* state either by popping an a entry (if there is one) or by pushing an a entry. We ignore bypasses in this viewpoint; if we pop an a entry while there are no b entries, then that a entry could have been treated as a bypass. An *A* state is reached from a *B* state only by popping the last a entry in a *B* state with a single a entry. Therefore, the generating function $A(a,x)$ satisfies

$$A(a,x)=1+\frac{x}{a}(A(a,x)-A(0,x))+aA(a,x)+xB(0,a,x).$$

The term 1 accounts for the start state. The term $(x/a)(A(a,x)-A(0,x))$ accounts for popping an a entry if there is one. The term $aA(a,x)$ accounts for pushing an a entry. Lastly, the term $xB(0,a,x)$ accounts for popping the final a from a *B* state with exactly one a entry, forcing all b entries to become a entries. It is important here that we assumed popping is always permitted in a *B* state.

We can reach a *B* state from an *A* state with at least one a by pushing a b. However, we do not want a term $(x/a)(A(a,x)-A(0,x))$ in the functional equation for $B(a,b,x)$ because the state resulting from pushing a single b does not allow for popping—this would violate our uniqueness conventions, because the entry that can be popped is the leftmost a entry which we could have popped before pushing the b entry. For this reason, we consider more elaborate transitions to *B* states: instead of pushing a single b entry, we push a sequence of b entries followed by at least one bypass (accounted for by the first term in $B(a,b,x)$ above) while a pop of an a entry may be followed by any number of bypasses (accounted for by the second term above). □

One can in principle iterate this functional equation starting with $A_0(a,x)=1$ and $B_0(a,b,x)=0$ to obtain terms of $A(0,x)$. It is clear from the description of pushing and popping that after $2n$ iterations the coefficient of each x^i for $0\le i\le n$ in the resulting $A_{2n}(0,x)$ will match the coefficient of x^i in $A(0,x)$. However, this is much slower than the dynamic programming approach.

After obtaining 5,000 terms of the sequence enumerating $\mathrm{Av}(4123,4231,4312)$ we apply the method of differential approximation [18], an experimental method that uses initial terms of a sequence to non-rigorously approximate asymptotic growth of the form

$$C\gamma^n n^{-1-\alpha}.$$

For this class, we predict $C\approx 0.01897$, $\gamma\approx 4.46841$, and $\alpha=-1$, so we conjecturally have the approximate asymptotics

$|\mathrm{Av}_n(4123,4231,4312)| \sim 0.01897 \cdot (4.46841)^n$.

We can approximate these constants to around 100 decimal places of accuracy.

A function $f(x)$ is said to be *differentially algebraic* if there exists some $k \geq 0$ and some polynomial $P(x_1, x_2, \ldots, x_{k+2})$ such that

$(\star)\ P(x, f(x), f'(x), \ldots, f^{(k)}(x)) = 0$

for all x. Equation $(\star)$ is called an *algebraic differential equation*. All algebraic and differentially finite (D-finite) generating functions are also differentially algebraic.

We have written a Maple program to use terms of a counting sequence to guess an algebraic differential equation which might be satisfied by the generating function of a given sequence. This program has not been able to guess an algebraic differential equation that might be satisfied by the generating function for $\mathrm{Av}(4123,4231,4312)$, despite searching through all possible algebraic differential equations

$P(x, f(x), f'(x), \ldots, f^{(k)}(x)) = 0$

with x_1-degree at most d, differential order at most k, and $(x_2, \ldots, x_{k+2})$-total degree at most m such that

$$(d+1)\binom{m+k+1}{m} + 5 \leq 5000.$$

In light of this, we make the following conjecture.

Conjecture 5.3

The generating function of the class $\mathrm{Av}(4123,4231,4312)$ *is not differentially algebraic.*

In particular, Conjecture 5.3 would imply that the generating function for this class is not D-finite. Note that every subclass of $\mathrm{Av}(123,231,312)$ has bounded enumeration, and thus by Theorem 4.6 their machines generate classes with algebraic generating functions. Thus it appears that the $\mathrm{Av}(123,231,312)$-machine is a minimal non-algebraic machine.

5.2. Three 2×4 classes

There are three so-far unenumerated 2×4 classes which may be generated by $\mathcal{C}$-machines. For full details on these three cases, we refer to the third author's thesis [29, Chapter 5], giving only an outline here. The containers of these machines are shown in Fig. 8. The first two containers, $\mathrm{Av}(123,231)$ and $\mathrm{Av}(123,231)$, may be analyzed using three parameters which record the sizes of the three blocks.

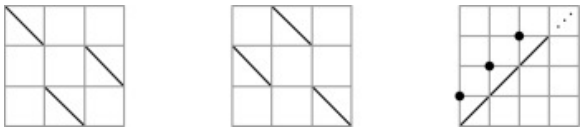


Fig. 8. From left to right, the classes $\mathrm{Av}(123,231)$, $\mathrm{Av}(123,312)$, and $\mathrm{Av}(231,321)$.

The class $\mathrm{Av}(231,321)$, however, is not a polynomial class; in fact, there are 2^{n-1} permutations of length n in this class. As such, the strategy of using a different variable to represent entries in each block is not effective. Instead, we label the entries of the maximum increasing final block (if there are any) by a, we label all entries in the rightmost sum component of the form $1 \ominus (12 \cdots \ell)$ by b. The remaining entries of the container are labeled by c. Fig. 9 shows three examples. As this assignment implies, we do not need to keep track of the actual shape formed by the c entries, even though they can take many different forms. Once an entry becomes a c entry, it stays a c entry until it is popped.

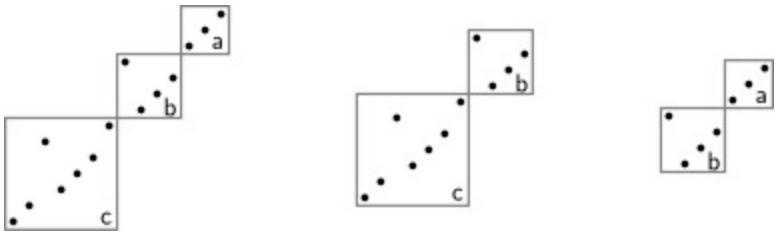


Fig. 9. Three examples of the variable assignments in the $\mathrm{Av}(231,321)$-machine.

With dynamic programming, we are able to easily compute 1,000 terms in each of first two enumerations, and 600 in the final case, and could extend this enumeration were it of interest. Again we use the method of differential approximation to fit these sequences to asymptotics of the form $C\gamma^n n^{-1-\alpha}$. The data for these three cases is collected in the table below.

Class	Sequence	$\gamma\approx$	$\alpha\approx$	OEIS [31]
$\mathrm{Av}(4123,4231)$	1,2,6,22,89,380,1677,7566	4.97689	−1	A165542
$\mathrm{Av}(4123,4312)$	1,2,6,22,89,382,1711,7922	$3+2\sqrt{2}$	1/2	A165545
$\mathrm{Av}(4231,4321)$	1,2,6,22,90,396,1837,8864	5.89249	−1	A053617

As in the previous subsection, we have also used these terms to attempt to conjecture differentiably algebraic generating functions fitting the known data, with no success. This even holds in the second example, despite the consistency of the conjectured values of γ and α with simple algebraic generating functions.

Conjecture 5.4

None of the classes $\mathrm{Av}(4123,4231)$, $\mathrm{Av}(4123,4312)$, *or* $\mathrm{Av}(4231,4321)$ *have differentiably algebraic generating functions.*

6. Concluding remarks

There are many more permutation classes that could be enumerated—either obtaining an explicit generating function or generating hundreds or thousands of terms—with $\mathcal{C}$-machines. While Section 4 initiates the study of the more general theory of how restrictions on a class $\mathcal{C}$ may imply certain properties of the class generated by the $\mathcal{C}$-machine, the four classes considered in Section 5 suggest that extending this classification may require great care.

For instance, we presented the class $\mathrm{Av}(4123,4231,4312)$ generated by the $\mathrm{Av}(123,231,312)$-machine. Recall that the class $\mathrm{Av}(123,231,312)$ is a polynomial class represented by the peg permutation 1−2− and that we conjecture that the class $\mathrm{Av}(4123,4231,4312)$ does not have a differentially algebraic generating function. One might suspect that the cause of this complicated behavior is the presence of two entries inflated by + or − in the peg permutation representing the class. However, there are (up to symmetry) four other two-cell machines, those represented by the peg permutations 1^+2^-, 2^+1^+, 2^+1^-, and 2^-1^+. The last three can be shown to generate Wilf-equivalent classes by considering their corresponding generation sequences, and all can be shown to generate classes whose generating functions are algebraic.

It appears that the $\mathrm{Av}(123,231,312)$-machine of Section 5 is harder to model than the other four two-cell machines for the same reason the kernel method fails to apply: when there is a single entry in the 1^- cell and at least one entry in the 2^- cell, the act of popping the leftmost entry causes all entries in the 2^- cell to shift downward into the 1^- cell. While we now know that the Noonan–Zeilberger Conjecture is false thanks to the

work of Garrabrant and Pak [17], among all potential concrete counterexamples, the class $\mathrm{Av}(4123,4231,4312)$ analyzed in Section 5.1 is simplest yet identified.

Acknowledgements

We are grateful to Mireille Bousquet-Mélou for suggesting a number of improvements to an earlier version of the paper, and to the referees for their careful reading of this work.

References

[1] M.H. Albert, M.D. Atkinson, M. Bouvel, N. Ruškuc, V. Vatter **Geometric grid classes of permutations** *Trans. Amer. Math. Soc.*, 365 (11) (2013), pp. 5859-5881

[2] M.H. Albert, M.D. Atkinson, N. Ruškuc **Regular closed sets of permutations** *Theoret. Comput. Sci.*, 306 (1–3) (2003), pp. 85-100

[3] M.H. Albert, S. Linton, N. Ruškuc **The insertion encoding of permutations** *Electron. J. Combin.*, 12 (1) (2005), Article 47 31 pp

[4] R. Arratia **On the Stanley–Wilf conjecture for the number of permutations avoiding a given pattern** *Electron. J. Combin.*, 6 (1999) Note 1, 4 pp

[5] M.D. Atkinson, M.M. Murphy, N. Ruškuc **Sorting with two ordered stacks in series** *Theoret. Comput. Sci.*, 289 (1) (2002), pp. 205-223

[6] J. Bloom, S. Elizalde **Pattern avoidance in matchings and partitions** *Electron. J. Combin.*, 20 (2) (2013), Article 5 38 pp

[7] J. Bloom, V. Vatter **Two vignettes on full rook placements** *Australas. J. Combin.*, 64 (1) (2016), pp. 77-87

[8] M. Bóna **Exact enumeration of 1342-avoiding permutations: a close link with labeled trees and planar maps** *J. Combin. Theory Ser. A*, 80 (2) (1997), pp. 257-272

[9] M. Bóna **The permutation classes equinumerous to the smooth class** *Electron. J. Combin.*, 5 (1998), Article 31 12 pp

[10] M. Bousquet-Mélou, A. Jehanne **Polynomial equations with one catalytic variable, algebraic series and map enumeration** *J. Combin. Theory Ser. B*, 96 (5) (2006), pp. 623-672

[11] T. Chow, J. West **Forbidden subsequences and Chebyshev polynomials** Discrete Math., 204 (1–3) (1999), pp. 119-128

[12] A.R. Conway, A.J. Guttmann **On the growth rate of 1324-avoiding permutations** *Adv. in Appl. Math.*, 64 (2015), pp. 50-69

[13] A.R. Conway, A.J. Guttmann, P. Zinn-Justin **1324-avoiding permutations revisited** arXiv:1709.01248 [math.CO]

[14] P.G. Doyle **Stackable and queueable permutations** arXiv:1201.6580 [math.CO]

[15] M. Droste, W. Kuich, H. Vogler (Eds.), *Handbook of Weighted Automata, Monographs in Theoretical Computer Science*, Springer-Verlag, Berlin, Germany (2009)

[16] M. Elder, V. Vatter **Problems and conjectures presented at the Third International Conference on Permutation Patterns, University of Florida, March 7–11, 2005** arXiv:math/0505504

[17] S. Garrabrant, I. Pak **Pattern avoidance is not P-recursive** arXiv:1505.06508 [math.CO]

[18] A.J. Guttmann, G.S. Joyce **On a new method of series analysis in lattice statistics** *J. Phys. A: Math. Gen.*, 5 (9) (1972), p. L81

[19] C. Homberger, V. Vatter **On the effective and automatic enumeration of polynomial permutation classes** *J. Symbolic Comput.*, 76 (2016), pp. 84-96

[20] S. Huczynska, V. Vatter **Grid classes and the Fibonacci dichotomy for restricted permutations** *Electron. J. Combin.*, 13 (2006), Article 54 14 pp

[21] F. Johansson, B. Nakamura **Using functional equations to enumerate 1324-avoiding permutations** *Adv. in Appl. Math*., 56 (2014), pp. 20-34
[22] T. Kaiser, M. Klazar **On growth rates of closed permutation classes** *Electron. J. Combin*., 9 (2) (2003), Article 10 20 pp
[23] D.E. Knuth ***The Art of Computer Programming, Vol. 1*** Addison-Wesley, Reading, Massachusetts (1968)
[24] D. Kremer **Permutations with forbidden subsequences and a generalized Schröder number** *Discrete Math.,* 218 (1–3) (2000), pp. 121-130
[25] D. Kremer **Postscript: "Permutations with forbidden subsequences and a generalized Schröder number"** *Discrete Math*., 270 (1–3) (2003), pp. 333-334
[26] D. Kremer, W.C. Shiu **Finite transition matrices for permutations avoiding pairs of length four patterns** *Discrete Math.,* 268 (1–3) (2003), pp. 171-183
[27] I. Le **Wilf classes of pairs of permutations of length 4** *Electron. J. Combin*., 12 (2005), Article 25 27 pp
[28] J. Noonan, D. Zeilberger **The enumeration of permutations with a prescribed number of "forbidden" patterns** *Adv. in Appl. Math*., 17 (4) (1996), pp. 381-407
[29] J. Pantone ***Structural Analysis of Permutation Classes*** PhD thesis University of Florida (2015)
[30] V.R. Pratt **Computing permutations with double-ended queues, parallel stacks and parallel queues** STOC '73: Proceedings of the Fifth Annual ACM Symposium on the Theory of Computing, ACM, New York (1973), pp. 268-277
[31] ***The On-line Encyclopedia of Integer Sequences (OEIS)*** published electronically at http://oeis.org/
[32] V. Vatter **Finitely labeled generating trees and restricted permutations** *J. Symbolic Comput*., 41 (5) (2006), pp. 559-572
[33] V. Vatter **Problems and conjectures presented at the problem session** S. Linton, N. Ruškuc, V. Vatter (Eds.), *Permutation Patterns, London Mathematical Society Lecture Note Series, vol. 376*, Cambridge University Press, Cambridge, England (2010), pp. 339-344
[34] V. Vatter **Small permutation classes** *Proc. Lond. Math. Soc*. (3), 103 (5) (2011), pp. 879-921
[35]V. Vatter **Permutation classes** M. Bóna (Ed.), *Handbook of Enumerative Combinatorics*, CRC Press, Boca Raton, Florida (2015), pp. 754-833
[36] H.S. Wilf **What is an answer?** *Amer. Math. Monthly*, 89 (5) (1982), pp. 289-292
[37] D. Zeilberger ***Enumerative and algebraic combinatorics*** T. Gowers (Ed.), Princeton Companion to Mathematics, Princeton University Press, Princeton, New Jersey (2008), pp. 550-561

[1]Pantone and Vatter were partially supported by the National Science Foundation under Grant Number DMS-1301692.

[2]In fact, the original inspiration for the kernel method came from Knuth's solution [23, Solution 2.2.1.4] to this very problem (though he did not use this language or the same functional equation).

[3]One must also add an additional equation $X\cdot(u1-u2)-1$ to force the distinctness of u1 and u2.

[4]Indeed, these classes fall under the purview not only of the rank encoding, but also of the finitely labeled generating trees of Vatter [32] and the insertion encoding of Albert, Linton, and Ruškuc [3].