Generating Permutations With Restricted Containers

Michael H. Albert
Department of Computer Science
University of Otago
Dunedin, New Zealand

Cheyne Homberger
Department of Mathematics & Statistics
University of Maryland, Baltimore County
Baltimore, Maryland

Jay Pantone*
Department of Mathematics
Dartmouth College
Hanover, New Hampshire

Nathaniel Shar
Department of Mathematics
Rutgers University
New Brunswick, New Jersey

Vincent Vatter*
Department of Mathematics
University of Florida
Gainesville, Florida

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 frequently be solved by either the kernel method or, much more easily, by guessing and checking. 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 enumerations, seem to not have D-finite generating functions.

1. Introduction

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* [22, 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).

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

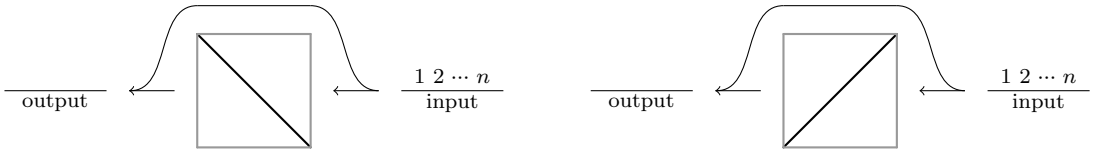


Figure 1: The Av(12)-machine, which generates Av(312), and the Av(21)-machine, which generates Av(321). The internal lines in diagrams of this type represent the allowed positions of the elements stored in the machine, so in the first case any entries in the machine must be decreasing when read left to right, and in the second case they must be increasing.

We are concerned here with a fairly general family of machines. Suppose that $\mathcal{C}$ is a permutation class (a downset in the classical permutation containment order). 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*).

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 Av(12)-machine, illustrated on the left of Figure 1. In this machine the container may only contain entries in decreasing order. Thus in generating permutations with the 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 Av(12)-machine is equivalent to a stack.

Before beginning the general study of $\mathcal{C}$-machines, we consider one more specific example, the Av(21)-machine, illustrated on the right of Figure 1. 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 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 clearly not possible. In the other direction,

we know that permutations in 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 Av(312) and Av(321) are both counted by the Catalan numbers, so the Av(21)- and Av(12)-machines generate equinumerous permutation classes (such classes are called *Wilf-equivalent*). This is no accident. Indeed, Section 2 shows how the description of Av(312) and 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 [11].)

For the rest of the introduction we review some basic definitions about permutation patterns and present a fundamental result describing classes generated by a $\mathcal{C}$-machine. 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) as 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. It follows readily that for any permutation class $\mathcal{C}$, the set of permutations that can be generated via the $\mathcal{C}$-machine also forms a permutation class.

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 functions. 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. It follows from Fekete's Lemma that this limit does exist for all classes generated by $\mathcal{C}$-machines (see Arratia [4]).

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.

Figure 2: The sum and skew sum operations

We now need to define the two operations on permutations illustrated in Figure 2. 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 Figure 2 is called the *skew sum*. We can now characterize the class of permutations which can be generated by a $\mathcal{C}$-machine.

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. Clearly 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 maxima) 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* [30, Question 4]. Atkinson, Murphy, and Ruškuc [5] showed that the permutation class sortable by two "ordered" stacks in series, despite having the infinite basis

$$\{2\ (2k-1)\ 4\ 1\ 6\ 3\ 8\ 5 \cdots (2k)\ (2k-3) : k \ge 2\},$$

is equinumerous to the class $\mathrm{Av}(1342)$, first counted by Bóna [7] (a simpler proof of this Wilf-equivalence result has since been given by Bloom and Vatter [6]). 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.

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. In some instances these functional equations can be solved either by the kernel method (in Section 2 and 3) or, much more easily, by what Gessel and Zeilberger [16] call "rigorous empirical evidence" (Section 4).

The reader will notice in Sections 3–6 that while we say that this 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 (in the best case) permits a solution or (in the second-best case) allows for the efficient generation of a large number of terms (for a ballpark figure, if this simplification step can be performed well then these methods can generate thousands of terms).

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 $\mathcal{C}$-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 6 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) but lacks a simple D-finite generating function, the second has an algebraic generating function (see Section 3 where we analyze it as a $\mathcal{C}$-machine), and the third seems very difficult to enumerate (the current record is 36 terms, computed by Conway and Guttmann [10] building on the approach of Johansson and Nakamura [19]).

Noonan and Zeilberger [27] 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 [13]) and Garrabrant and Pak [15] 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. The Catalan Classes & Uniqueness of Generation Sequences

We now introduce our uniqueness conventions and show how they implicitly define a length-preserving bijection between the classes $\mathrm{Av}(312)$ and $\mathrm{Av}(321)$. While this ground is well-trodden, we believe that at the very least the $\mathcal{C}$-machine perspective presents a particularly straight-forward view of this Wilf-equivalence.

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

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

(U2) all left-to-right maxima should bypass the container.

Note that the rules (U1) and (U2) correspond to us 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 2.1. *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.*

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 $\pi(i)$ 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 and its placement relative to the other entries in the container is uniquely determined by its position in π. □

These rules implicitly give a bijection between Av(312) and Av(321). When generating a permutation with either the Av(12)- or Av(21)-machine, we must pop whenever possible and all left-to-right maxima must bypass the container. Moreover, whenever we push into the container, there is a unique place for the new entry to be placed. In fact, this argument establishes that there is a bijection between Av(312) and Av(321) that preserves the locations and values of left-to-right maxima. (This bijection is equivalent, by symmetry, to one presented by Knuth [22].)

Note that this bijection also restricts to a bijection between permutations that can be generated by the $\mathrm{Av}(12, k\cdots21)$- and $\mathrm{Av}(21, 12\cdots k)$-machines, implying that the classes $\mathrm{Av}(312, (k+1)\dots21)$ and $\mathrm{Av}(321, (k+1)12\dots k)$ are Wilf-equivalent. This result was first established by Chow and West [9], where they 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 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 Figure 3.

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

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

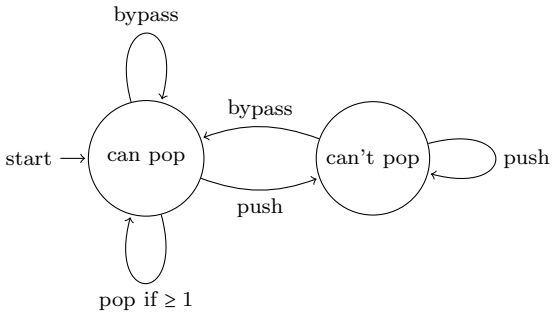


Figure 3: An automaton representing the Av(21)-machine

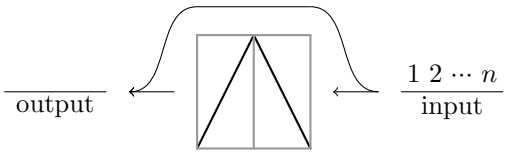


Figure 4: The Av(312, 213)-machine generates a Schröder class.

To solve this system with the kernel method[2] we first solve for $g(x,u)$ in terms of $f(x,u)$,

$$g(x,u) = \frac{u}{1-u} f(x,u),$$

and then substitute this into the first equation and collect $f(x,u)$ terms, leaving

$$\left(1 - x - \frac{xu}{1-u} - \frac{x}{u}\right) f(x,u) = 1 - \frac{x}{u} f(x,0).$$

Finally, we set $u = (1-\sqrt{1-4x})/2$, and find that $f(x,0) = (1-\sqrt{1-4x})/2x$.

3. 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 [8, 23, 24, 25, 26], it has been shown that these 56 symmetry classes fall into 38 Wilf equivalence classes, of explicit generating functions have been found for all but 9. By far the largest of these Wilf equivalence classes consists of 10 symmetry classes enumerated by the large Schröder numbers (this Wilf equivalence class was found by Kremer [23, 24]). 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 Av(4312, 4213), which is generated by the Av(312, 213)-machine shown in Figure 4. As indicated by the dark lines in this figure, permutations in Av(312, 213) consist of an increasing sequence followed by a decreasing subsequence.

[2]In fact, the original inspiration for the kernel method came from Knuth's solution [22, Solution 2.2.1.4] to this very problem (though he did not use this language or the same functional equation).

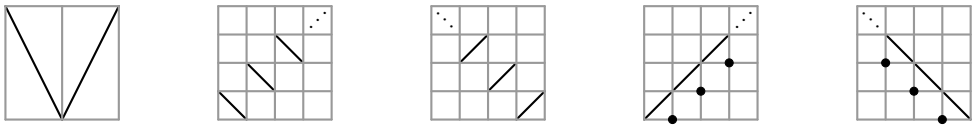


Figure 5: Five classes whose machines generate Schröder classes

By (U1) and (U2), pops and bypasses in the Av(312, 213)-machine function the same as they do in the 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 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. However, this is the last occasion where our functional equations are simple enough to apply this method, and in Section 4 we introduce a different method.

First we solve for $g(x,u)$ in terms of $f(x,u)$ and $f(x,0)$:

$$g(x,u) = \frac{2uf(x,u) - uf(x,0)}{1-2u}.$$

Next we substitute this into the $f(x,u)$ equation and collect $f(x,u)$ terms, leaving

$$\left(1 - x - \frac{2xu}{1-2u} - \frac{x}{u}\right) f(x,u) = 1 - \left(\frac{xu}{1-2u} + \frac{x}{u}\right) f(x,0).$$

Finally, we make the substitution $u = (1 + x - \sqrt{1-6x+x^2})/4$ (the generating function for the small Schröder numbers) to make the left-hand side 0, and solve to find that

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

as expected.

Recall that Proposition 2.1 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). We have observed above that in the case of the Av(312, 213)-machine there are two different types of push operations. Therefore, just as in the previous section, if we can find other classes $\mathcal{C}$ for which there are two different push operations whenever the container is nonempty, the class generated by the $\mathcal{C}$-machine will not only be counted by the Schröder numbers, but there will be bijections between all such classes which preserve the positions and values of left-to-right maxima. Figure 5 shows five such

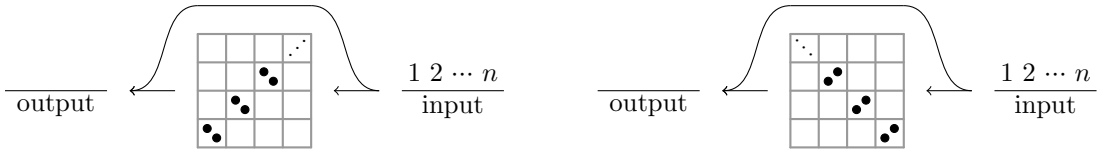


Figure 6: The $\mathcal{F}_\oplus$- and $\mathcal{F}_\ominus$-machines

classes whose associated machines generate Schröder classes, each of which has two types of allowable pushes into non-empty containers.

To conclude this section we observe that there cannot be bijections that preserve the positions and values of left-to-right maxima between these six Schröder classes and the other four Schröder classes. This is because each of the other four Schröder classes has at least one basis element of length four that does not begin with 4.

4. The Fibonacci Machines and the Method of Guess and Check

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 Figure 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 $F_0 = F_1 = 1$.

At first glance it might seem that the $\mathcal{F}_\oplus$- and $\mathcal{F}_\ominus$-machines should generate Wilf-equivalent classes, yet for $n \geq 7$ the $\mathcal{F}_\ominus$-machine generates strictly more permutations than the $\mathcal{F}_\oplus$-machine. To give a concrete example of why this is the case, 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 $\mathcal{F}_\ominus = \mathrm{Av}(123, 132, 213)$. Hence by Thoerem 1.1, 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.

To enumerate the permutations generated by the $\mathcal{F}_\oplus$-machine, we employ the guess and check methodology as outlined by Gessel and Zeilberger [16]. We derive functional equations satisfied by the generating function of the class under investigation. Separately, we use a dynamic programming approach to find many terms of the enumeration of the class, so that we may algorithmically *guess* the generating function we seek. Lastly, we substitute this guess into the functional equations and *check* that the functional equations are indeed satisfied. Given that there is a unique power series solution to each of our sets of functional equations, this confirmation step ensures that the result is fully rigorous.

4.1 Enumerating the $\mathcal{F}_{\oplus}$-Machine

We start by crafting an automaton to represent the $\mathcal{F}_{\oplus}$-machine, similar to that in Figure 3 for the 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 Figure 7, has 5 states: E represents an empty machine, S_p and S_n represent states in which the rightmost layer has only one entry and pops are permitted or forbidden, respectively, and D_p and D_n represent states in which the rightmost layer has two entries and pops are permitted or forbidden, respectively.

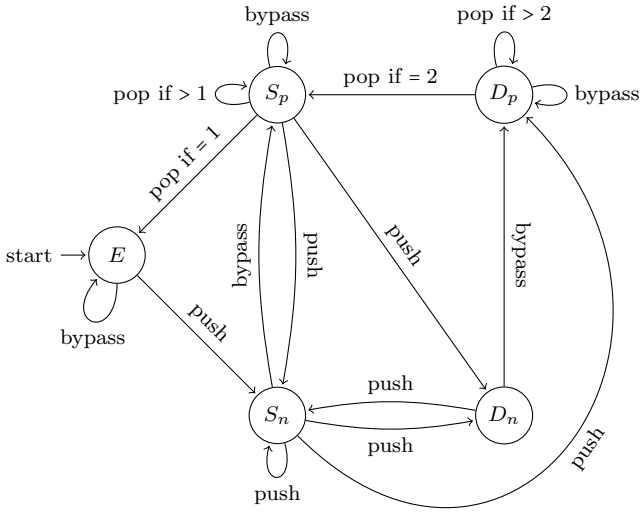


Figure 7: An automaton representing the $\mathcal{F}_{\oplus}$-machine

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 Figure 7 translates to the following system of functional equations.

$$\begin{aligned}
E &= 1 + xE + x\left(S_p\big|_{u=0}\right)\\
S_p &= x(S_n + S_p) + \frac{x}{u}\left(S_p - S_p\big|_{u=0}\right) + x\left(D_p\big|_{u=0}\right)\\
S_n &= E + u(S_n + S_p) + u^2(D_n + D_p)\\
D_p &= x(D_n + D_p) + \frac{x}{u}\left(D_p - D_p\big|_{u=0}\right)\\
D_n &= S_n + S_p
\end{aligned}$$

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.

While it it possible to use the kernel method to solve for $E(x)$, it is easier to guess and check a solution. First, using dynamic programming we can compute the first 30 terms of the generating functions $S_p(x,0)$ and $D_p(x,0)$ (indeed, one can even obtain 30 terms by iterating the system of functional equations above). Based on these terms, we then guess that $S_p(x,0)$ satisfies

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

while $D_p(x,0)$ satisfies

$$\begin{aligned}(2x^5+8x^4-x^3)D_p(x,0)^4-(x^5+3x^4-23x^3+4x^2)D_p(x,0)^3\\ +(2x^4-4x^3+20x^2-4x)D_p(x,0)^2\\ -(x^3-4x^2-4x+1)D_p(x,0)\\ +x=0.\end{aligned}$$

Substituting this information back into our system of functional equations yields a system of 5 equations with 5 unknowns. (Maple is perfectly happy to perform the algebraic manipulation when $S_p(x,0)$ and $D_p(x,0)$ are defined as "`RootOf`" expressions.) Solving this system of 5 equations with 5 unknowns yields minimal polynomials satisfied by $E(x)$, $S_p(x,u)$, $S_n(x,u)$, $D_p(x,u)$, and $D_n(x,u)$. From this we can check that $S_p(x,0)$ and $D_p(x,0)$ are indeed equal to the previously guessed equations.

It is clear from the system of equations that there is exactly one set of formal power series solutions. Unfortunately, it is not clear whether the solution that Maple found is that solution. We verify in Appendix A that the solutions provided by Maple are indeed analytic at the origin, and therefore they are the desired solutions. The solutions are now rigorous and we see that $E(x)$ (the generating function for the class generated by the $\mathcal{F}_\oplus$-machine) satisfies

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

The enumeration of this class is given by sequence A257561 in the OEIS [28].

In this case, Maple can explicitly solve for $E(x)$ and inspection reveals that the singularity closest to the origin is

$$\frac{7+3\sqrt{5}}{2}-\sqrt{22+10\sqrt{5}}\;\approx\;0.1937,$$

and thus the growth rate of this permutation class is

$$\frac{\left(3-\sqrt{5}\right)\left(7+3\sqrt{5}+2\sqrt{22+10\sqrt{5}}\right)}{4}\;\approx\;5.1621.$$

4.2 Enumerating 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).

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.

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

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 state in which popping is legal (e.g., $W_n \longrightarrow xW_p$).

The remainder of the 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.

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

Again, Maple solves this system of 5 equations with 5 unknowns, but does not immediatly give the minimal polynomial that s satifies. To find this, we use the `GroebnerBasis` package, and find that

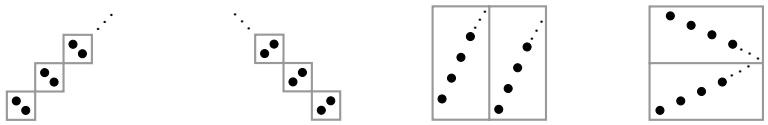


Figure 8: The permutations shown, and all of their symmetries, are the only obstructions that prevent a permutation class from being a polynomial class.

s satisfies

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

Therefore, the generating function of the class generated by the $\mathcal{F}_\ominus$-machine is $s(x)$. This implies that the growth rate of this class is

$$\frac{67240 + (779\sqrt{57} - 1927)(1502 + 342\sqrt{57})^{1/3} - (19\sqrt{57} - 457)(1502 + 342\sqrt{57})^{2/3}}{40344} \approx 5.219.$$

The enumeration of the class is given by sequence A106228 in the OEIS [28].

5. Finite, Bounded, and Polynomial Machines

As the reader may have noticed, the analysis of $\mathcal{C}$-machines typically depends on the very specific structure of $\mathcal{C}$ itself. In this section we explore three families of permutation classes—finite, bounded, and polynomial classes—for which we are able to establish general results.

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 $\pi(i)$ is the number of entries below it and to its right (technically, their definition of rank is 1 more than this). 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[3].

Theorem 5.1. *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.*

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 [20] 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 nth 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 [18] reproved the Fibonacci Dichotomy using what are known as grid classes and gave an explicit characterization of these classes. These results are collected below.

[3]Indeed, these classes fall under the purview not only of the rank encoding, but also of the finitely labeled generating trees of Vatter [29] and the insertion encoding of Albert, Linton, and Ruškuc [3].

Theorem 5.2 (Kaiser and Klazar [20] and Huczynska and Vatter [18])**.** *For a permutation class $\mathcal{C}$ the following are equivalent:*

(1) $|\mathcal{C}_n|$ is given by a polynomial for all sufficiently large n,

(2) $|\mathcal{C}_n| < F_n$ for some n,

(3) $\mathcal{C}$ does not contain arbitrary long permutations of any of the forms shown in Figure 8 (or any symmetries of those).

While we do not need to appeal to the characterization above, we do require following fact that follows from it.

Proposition 5.3. *Every polynomial class is finitely based.*

Proposition 5.3 is explicitly proved in the conclusion of Huczynska and Vatter [18] and also follows from the later and more general Vatter [31, Theorem 6.2].

Our result about polynomial classes requires one further notion. Inspired by Wilf's infuential *Monthly* article "What is an answer?" [32], Zeilberger [33] defined a *Wilfian formula* for the sequence $\{a_n\}$ to be a polynomial-time (in n) algorithm that computes a_n. 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 5.4. *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 $|\mathcal{C}_n| \le c(n)$ for all n. By Proposition 5.3, $\mathcal{C} = \mathrm{Av}(B)$ 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 empty "can pop" state that consist 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. An example of this, the $\mathrm{Av}(231, 321)$-machine, is analyzed in the next section. The permutations in the class $\mathrm{Av}(231, 321)$ are all direct sums of permutations of the form $k12\cdots(k-1)$. This is of note because such classes may no longer be polynomial; the class $\mathrm{Av}(231, 321)$ has an growth rate of 2.

Needless to say, the algorithm described in the proof of Theorem 5.4 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 the next section.

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 the bounded classes are a special case of the 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 [17].

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 $\bullet$, 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 proof of Theorem 5.2, Huczynska and Vatter [18] 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 5.5. *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 $\mathbb{N}^m$ (and thus also on $\mathbb{P}^m$) 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 $\bullet$ 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 5.6 (Homberger and Vatter [17])**.** *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*

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

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 their accepting languages have algebraic generating functions. (See Droste, Kuich, and Vogler [12, Chapter 7].)

Theorem 5.7. *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 5.6. 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 5.1) 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 $\mathbb{P}^{|\tilde{\rho}|}$. 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

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

I.e., 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. □

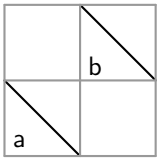
As with all the results of this section, note that Theorem 5.7 represents only a sufficient condition for algebraicity. In particular, it does not apply to any of the Schröder machines which nevertheless generate classes with algebraic generating functions.

6. 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 5.4 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.

6.1 $\mathrm{Av}(4123, 4231, 4312)$

By Theorem 1.1, 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.



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.

Thus 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 `true` or `false`, depending on whether popping is allowed. It is not difficult to see that the following are the transition rules for this machine

(assume that $a, b \geq 1$ unless stated otherwise):

$$\begin{aligned}
(0,0,\mathtt{T}) &\longrightarrow \{(1,0,\mathtt{F}),(0,0,\mathtt{T})\},\\
(a,0,\mathtt{F}) &\longrightarrow \{(a+1,0,\mathtt{F}),(a,1,\mathtt{F}),(a,0,\mathtt{T})\},\\
(a,0,\mathtt{T}) &\longrightarrow \{(a+1,0,\mathtt{F}),(a,1,\mathtt{F}),(a,0,\mathtt{T}),(a-1,0,\mathtt{T})\},\\
(a,b,\mathtt{F}) &\longrightarrow \{(a,b+1,\mathtt{F}),(a,b,\mathtt{T})\},\\
(a,b,\mathtt{T}) &\longrightarrow \{(a,b+1,\mathtt{F}),(a,b,\mathtt{T}),(a-1,b,\mathtt{T})\} \quad (\text{for } a \geq 2),\\
(1,b,\mathtt{T}) &\longrightarrow \{(1,b+1,\mathtt{F}),(1,b,\mathtt{T}),(b,0,\mathtt{T})\}.
\end{aligned}$$

The start state is $(0,0,\mathtt{T})$ and transitions of the form $(a,b,\mathtt{F}) \longrightarrow (a,b,\mathtt{T})$ and $(a,b,\mathtt{T}) \longrightarrow (a,b,\mathtt{T})$ correspond to performing bypasses. 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 classes is sequence A257562 in the OEIS [28].

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 labelled 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 labelled a, ℓ entries labelled 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.

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 $b(A(a,x) - A(0,x))$ 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 $B(a,b,x)$ below) while a pop of an a entry may be followed by any number of bypasses (accounted for by the second term below).

Therefore, the following functional equations are derived:

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

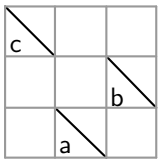
Here the a in the denominator in the first term in $B(a,b,x)$ accounts for the fact that $B(a,b,x)$ tracks one fewer than the number of $A(a,x)$.

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.

Note that every subclass of $\mathrm{Av}(123,231,312)$ has bounded enumeration, and thus by Theorem 5.7 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.

6.2 $\mathrm{Av}(4123,4231)$

The class $\mathrm{Av}(4123,4231)$ is generated by the $\mathrm{Av}(123,231)$-machine, and the standard figure of $\mathrm{Av}(123,231)$ is shown below.



We can represent the states of this machine with 4-tuples of the form (a,b,c,P) that record the number of a, b, and c entries together with whether pops are permitted. For uniqueness, we always choose the tuple with a as large as possible. The transitions are largely analogous to the previous case, with the addition that when there are both a and b entries, one can perform a stack-like sequences of pushes and pops using c entries. Using dynamic programming we are able to compute the first 1,000 terms of the enumeration of $\mathrm{Av}(4123,4231)$, sequence A165542 in the OEIS [28].

As in the previous case, we use slightly complicated transitions to B states to ensure that at the end of every such transition popping is always allowed. Rather than reaching a B state from an A state by just pushing a b entry to an A state with at least one a entry, we instead allow pushing an arbitrary number of b entries (at least one), then performing a nonempty sequence of pushes and pops of c entries. We observe that the sequence of pushes and pops of c entries is essentially a sequence of pushing and popping entries in and out of a stack, and so the number of different ways to do this with n pushes and n pops is the nth Catalan number — moreover, a push and immediate pop of a c entry takes the place of a bypass. To this end, define

$$C(x) = \frac{1 - 2x - \sqrt{1-4x}}{2x}.$$

Now, the generating function for the number of ways to take an A state, push some number of b entries, then perform a stack-like operation sequence on c entries is $(A(a,x)-A(0,x))(bC(x)/(1-b))$, except we must account for the fact that B tracks *one fewer* than the number of a entries. Therefore, these transitions are accounted for by a term

$$(A(a,x)-A(0,x))\frac{bC(x)}{a(1-b)}.$$

There are two ways to transition from one B state to another B state. The first is to push a b entry to a B state. However, as before, in order to leave the machine in a state where popping is allowed, we push an arbitrary number of b entries (at least one) and then perform a stack-like operation sequence on c entries. These transitions are accounted for by a term

$$B(a,b,x)\frac{bC(x)}{1-b}.$$

Lastly, we may pop an a entry from a B state, and so long as there are at least two a entries the new state will still be a B state. The generating function for B states with at least two a entries is $B(a,b,x)-B(0,b,x)$. After popping an a entry, we may choose whether or not to perform a stack-like operations sequence on c entries. Accordingly, these transitions are represented by

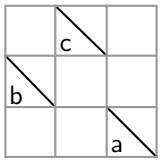
$$\frac{x}{a}(1+C(x))(B(a,b,x)-B(0,b,x)).$$

Combining, we obtain the pair of functional equations

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

6.3 Av(4123, 4312)

The class Av(4123, 4312) is generated by the Av(123, 312)-machine. The standard figure of the class Av(123, 312) is shown below. Despite its obvious Wilf-equivalence to Av(123, 231), the classes generated by these two machines are not Wilf-equivalent.



We represent the current state of the Av(123, 312)-machine by a 4-tuple (a,b,c,P) exactly as in the previous case.

Due to the fact that a b entry is never directly pushed into the machine, the transitions between states in the Av(123, 312)-machine are subtly different from those of the Av(123, 231)-machine. Consider a state of the Av(123, 312)-machine that contains precisely k entries, all labelled a. The

next entry pushed into the machine can be placed horizontally between any two a entries, to the left of the leftmost a entry, or to the right of the rightmost a entry — $k+1$ locations in total. When the new entry is pushed to the left of the leftmost a entry, the new entry becomes an a entry. However, pushing the new entry in any other position converts all a entries to its left to become b entries, while the new entry becomes a c entry. For example, one possible state transition is

$$(7,0,0,\mathtt{F}) \longrightarrow (2,5,1,\mathtt{F}).$$

A similar phenomenon appears when popping entries. Once the a entries have been split so that there are now b and c entries, there are three valid operations: bypass, push a c entry, or pop a b entry. We can neither pop a c entry nor push a b entry. Moreover, when the last b entry has been popped, the entries in the machine form a decreasing sequence and therefore all become a entries. A possible state transition illustrating this effect is

$$(4,1,3,\mathtt{T}) \longrightarrow (7,0,0,\mathtt{T}).$$

There are 6 total transition rules for this machine:

1. $(0,0,0,\mathtt{T}) \longrightarrow \{(1,0,0,\mathtt{F}),(0,0,0,\mathtt{T})\}$
2. $(a,0,0,\mathtt{F}) \longrightarrow \{(a+1,0,0,\mathtt{F}),(a,0,0,\mathtt{T})\} \cup \{(i,a-i,1) : 0 \le i \le a-1\}$
3. $(a,0,0,\mathtt{T}) \longrightarrow \{(a+1,0,0,\mathtt{F}),(a,0,0,\mathtt{T}),(a-1,0,0,\mathtt{T})\} \cup \{(i,a-i,1) : 0 \le i \le a-1\}$
4. $(a,1,c,\mathtt{T}) \longrightarrow \{(a,1,c+1,\mathtt{F}),(a,1,c,\mathtt{T}),(a+c,0,0,\mathtt{T})\}$
5. $(a,b,c,\mathtt{F}) \longrightarrow \{(a,b,c+1,\mathtt{F}),(a,b,c,\mathtt{T})\}$
6. $(a,b,c,\mathtt{T}) \longrightarrow \{(a,b,c+1,\mathtt{F}),(a,b,c,\mathtt{T}),(a,b-1,c,\mathtt{T})\}, \quad (b \ge 2)$

As before these transition rules can be implemented with dynamic programming to compute the first 1,000 terms of the enumeration of $\mathrm{Av}(4123,4312)$, sequence A165545 in the OEIS [28].

One can construct functional equations to model the $\mathrm{Av}(123,312)$-machine using a framework similar to the $\mathrm{Av}(123,231)$-machine considered above. We say that a machine state is an A state if it has no b or c entries, and a B state otherwise. Note that our transition rules imply that there is at least one b entry if and only if there is at least one c entry.

Let $A(a,x)$ be the generating function counting states for which the coefficient of $a^k x^n$ is the number of states that contain k entries labelled a, no b or c entries, and for which n entries have been output thus far. Let $B(a,b,c,x)$ be the generating function counting states for which the coefficient of $a^k b^\ell c^m x^n$ is the number of states that contain k entries labelled a, $\ell+1$ entries labelled b, $m>0$ entries labelled c, and for which n entries have been output thus far. Similar to the previous case, B tracks *one fewer* than the number of b entries, so that $B(a,0,c,x)$ is the generating function for states with exactly one b entry.

An A state can be reached from an A state either by popping an a entry (if there is one) or by pushing an a entry. An A state can be reached from a B state with exactly one b entry by popping this b entry, in which case all c entries now become a entries. Therefore, we have

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

For the B states, we seek a functional equation in which each term represents not a single operation but a sequence of operations that leaves us in a state in which we are allowed to pop. There are two such sequences from B states to B states: the first involves pushing any positive number of c entries and then performing any positive number of bypasses, while the second involves popping a b entry (if there are at least two), and then performing any positive number of bypasses. These two operation sequences are counted by

$$\frac{cx}{(1-c)(1-x)}B(a,b,c,x)$$

and

$$\frac{x}{b(1-x)}(B(a,b,c,x)-B(a,0,c,x)),$$

respectively.

The only way to transition from an A state to a B state is to push a c entry into one of k rightmost positions in an A state with k entries labelled a (pushing to the one other location creates an A state with $k+1$ entries labelled a). As before, we follow this with pushing some positive number of c entries and performing some positive number of bypasses. To illustrate, the operation of splitting k entries labelled a into i entries labelled a and $k-i$ entries labelled b for any $0 \le i \le k-1$ turns a term $C_{k,n}a^kx^n$ in $A(a,x)$ into a term

$$C_{k,n}(b^{k-1}+ab^{k-2}+a^2b^{k-3}+\cdots+a^{k-3}b^2+a^{k-2}b+a^{k-1})x^n = C_{k,n}\frac{a^k-b^k}{a-b}x^n$$

in $B(a,b,c,x)$ (keeping in mind that $B(a,b,c,x)$ tracks one fewer than the number of b entries). Therefore, these transitions are represented by the term

$$\frac{cx}{(1-c)(1-x)}\left(\frac{A(a,x)-A(b,x)}{a-b}\right).$$

Combining, we obtain the set of functional equations

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

6.4 $\mathrm{Av}(4231,4321)$

Our final example is the class $\mathrm{Av}(4231,4321)$, which is generated by the $\mathrm{Av}(231,321)$-machine. Every permutation in $\mathrm{Av}(231,321)$ can be expressed as a sum $\tau_1 \oplus \cdots \oplus \tau_k$ where each τ_i is either the permutation 1 or a permutation of the form $1 \ominus (12\cdots\ell)$. This class is shown below. It is clearly a symmetry of the two rightmost classes in Figure 5, despite the fact that the machines corresponding to those two classes generate Schröder classes and the machine corresponding to $\mathrm{Av}(231,321)$ does not.

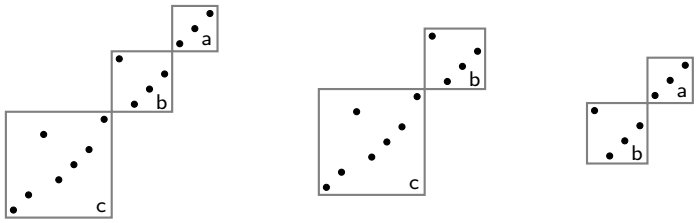
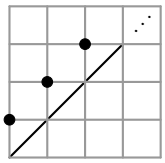


Figure 9: Three examples of the variable assignments in the $\mathrm{Av}(231, 321)$-machine.



Unlike all previous machines in the section, the class $\mathrm{Av}(231, 321)$ is not a polynomial class; in fact, $|\mathrm{Av}_n(231, 321)| = 2^{n-1}$. As such, the strategy of using a different variable to represent entries in each monotone component will no longer work. Instead, we call any entries in a rightmost monotone component (if it exists) a entries, any entries in the rightmost sum component of the form $1 \ominus (12\cdots\ell)$ entries labelled b, and all other entries c entries. Figure 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.

When there are no c entries and a b entry is popped, all of the remaining entries become a entries (as they now form an increasing sequence). When there are only a entries (say, k of them), a new maximum entry may be pushed in $k+1$ places. Inserting the new maximum in the rightmost location creates a longer sequence of a entries. All other options create some combination of b and c entries. A similar phenomenon occurs when pushing a new maximum when there is at least one c entry; here, the a entries become some combination of b entries and c entries, and all old b entries become c entries.

Let (a, b, c, P) represent the state of the machine that has a entries of type a, etc., and P is either T or F depending on whether popping is allowed. Assume $a, b, c \geq 1$ unless otherwise stated. The transition rules are as follows.

1. $(0,0,0,\mathtt{T}) \longrightarrow \{(1,0,0,\mathtt{F}), (0,0,0,\mathtt{T})\}$
2. $(a,0,0,\mathtt{F}) \longrightarrow \{(a+1,0,0,\mathtt{F}), (a,0,0,\mathtt{T})\} \cup \{(0, a-i+1, i, \mathtt{F}) : 0 \leq i \leq a-1\}$
3. $(a,0,0,\mathtt{T}) \longrightarrow \{(a+1,0,0,\mathtt{F}), (a,0,0,\mathtt{T}), (a-1,0,0,\mathtt{T})\} \cup \{(0, a-i+1, i, \mathtt{F}) : 0 \leq i \leq a-1\}$
4. $(0,b,0,\mathtt{F}) \longrightarrow \{(1,b,0,\mathtt{F}), (0,b,0,\mathtt{T})\}$
5. $(0,b,0,\mathtt{T}) \longrightarrow \{(1,b,0,\mathtt{F}), (0,b,0,\mathtt{T}), (b-1,0,0,\mathtt{T})\}$
6. $(a,b,0,\mathtt{F}) \longrightarrow \{(a+1,b,0,\mathtt{F}), (a,b,0,\mathtt{T})\} \cup \{(0, a-i+1, b+i, \mathtt{F}) : 0 \leq i \leq a-1\}$
7. $(a,b,0,\mathtt{T}) \longrightarrow \{(a+1,b,0,\mathtt{F}), (a,b,0,\mathtt{T}), (a+b-1,0,0,\mathtt{T})\} \cup \{(0, a-i+1, b+i, \mathtt{F}) : 0 \leq i \leq a-1\}$
8. $(0,b,c,\mathtt{F}) \longrightarrow \{(1,b,c,\mathtt{F}), (0,b,c,\mathtt{T})\}$

9. $(0,b,c,\mathtt{T}) \longrightarrow \{(1,b,c,\mathtt{F}),(0,b,c,\mathtt{T}),(0,b,c-1,\mathtt{T})\}$
10. $(a,b,c,\mathtt{F}) \longrightarrow \{(a+1,b,c,\mathtt{F}),(a,b,c,\mathtt{T})\} \cup \{(0,a-i+1,b+c+i,\mathtt{F}) : 0 \le i \le a-1\}$
11. $(a,b,c,\mathtt{T}) \longrightarrow \{(a+1,b,c,\mathtt{F}),(a,b,c,\mathtt{T}),(a,b,c-1,\mathtt{T})\} \cup \{(0,a-i+1,b+c+i,\mathtt{F}) : 0 \le i \le a-1\}$

By implementing these transition rules with dynamic programming we were able to compute the first 600 terms of the enumeration of $\mathrm{Av}(4231,4321)$, sequence A053617 in the OEIS [28]. (The computation of terms in this case is more resource intensive than the other examples due to the larger number of transitions possible at each step.)

Unlike the previous cases where we were able to cleverly avoid the "can pop" / "can't pop" model, we revert to a setup similar to that used to enumerate the $\mathcal{F}_{\oplus}$-machine. We say that a state is an A state if there are only a entries and no b or c entries, and a B state otherwise. We further split A states into A_p and A_n states depending on whether popping is or is not permitted, respectively, and we analogously split B states into B_p and B_n states.

The transition rules above translate almost directly into the functional equations below.

$$\begin{aligned}
A_p &= 1 + x(A_p(a,x) + A_n(a,x)) + \frac{x}{a}(A_p(a,x) - A_p(0,x)) + \frac{x}{a}B_p(a,a,0,x),\\
A_n &= a(A_p(a,x) + A_n(a,x)),\\
B_p &= x(B_p(a,b,c,x) + B_n(a,b,c,x)) + \frac{x}{c}(B_p(a,b,c,x) - B_p(a,b,0,x)),\\
B_n &= a(B_p(a,b,c,x) + B_n(a,b,c,x)) + \frac{b^2}{c-b}((A_p(c,x) - A_p(b,x)) + (A_n(c,x) - A_n(b,x)))\\
&\quad + \frac{b^2}{c-b}((B_p(c,c,c,x) - B_p(b,c,c,x)) + (B_n(c,c,c,x) - B_n(b,c,c,x))).
\end{aligned}$$

6.5 Guessing Generating Functions

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

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

for all x. Equation $(\star)$ is called an *algebraic differential equation*. All algebraic and differentially finite (D-finite) generating functions are also differentially algebraic. For an example of a differentially algebraic series that is neither algebraic nor differentially finite consider the Bell numbers, whose exponential generating function $B(x)$ satisfies

$$B(x)B'(x) - B(x)B''(x) + B'(x)^2 = 0.$$

(Klazar [21] proves that the ordinary generating function for the Bell numbers is not 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 algebraic differential equations that might be satisfied by any of the generating functions considered in this section. In light of this, we make the following conjecture.

Conjecture 6.1. *None of the generating functions for the classes* $\mathrm{Av}(4123, 4231, 4312)$*,* $\mathrm{Av}(4123, 4231)$*,* $\mathrm{Av}(4123, 4312)$*, or* $\mathrm{Av}(4231, 4321)$ *are differentially algebraic. (In particular, none of these classes have D-finite generating functions.)*

7. Concluding Remarks

There are of course many more permutation classes that can be enumerated—either obtaining an explicit generating function or generating hundreds of terms—with $\mathcal{C}$-machines. While Section 5 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 studied in Section 6 suggest that extending this classification may take great care.

For example, 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 four can be shown to generate classes whose generating functions are algebraic.

It appears that the $\mathrm{Av}(123, 231, 312)$-machine of Section 6 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 [15], among all potential concrete counterexamples, the class $\mathrm{Av}(4123, 4231, 4312)$ analyzed in Section 6.1 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 in particular for pointing out the need to provide the analytic argument in the appendix and also to user Fan Zheng on MathOverflow whose comment on a question we posed there provided the inspiration for the aforementioned analytic argument.

References

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

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

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

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

[5] ATKINSON, M. D., MURPHY, M. M., AND RUŠKUC, N. Sorting with two ordered stacks in series. *Theoret. Comput. Sci. 289*, 1 (2002), 205–223.

[6] BLOOM, J., AND VATTER, V. Two vignettes on full rook placements. *Australas. J. Combin. 64*, 1 (2016), 77–87.

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

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

[9] CHOW, T., AND WEST, J. Forbidden subsequences and Chebyshev polynomials. *Discrete Math. 204*, 1-3 (1999), 119–128.

[10] CONWAY, A. R., AND GUTTMANN, A. J. On the growth rate of 1324-avoiding permutations. *Adv. in Appl. Math. 64* (2015), 50–69.

[11] DOYLE, P. G. Stackable and queueable permutations. arXiv:1201.6580 [math.CO].

[12] DROSTE, M., KUICH, W., AND VOGLER, H., Eds. *Handbook of weighted automata*. Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, 2009.

[13] ELDER, M., AND VATTER, V. Problems and conjectures presented at the Third International Conference on Permutation Patterns, University of Florida, March 7–11, 2005. arXiv:math/0505504 [math.CO].

[14] FISCHER, G. *Plane algebraic curves*, vol. 15 of *Student Mathematical Library*. American Mathematical Society, Providence, RI, 2001. Translated from the 1994 German original by Leslie Kay.

[15] GARRABRANT, S., AND PAK, I. Pattern avoidance is not P-recursive. arXiv:1505.06508 [math.CO].

[16] GESSEL, I. M., AND ZEILBERGER, D. An empirical method for solving (rigorously!) algebraic functional equations of the form $f(p(x,t),p(x,1),x,t)=0$. arXiv:1412.8360 [math.CO].

[17] HOMBERGER, C., AND VATTER, V. On the effective and automatic enumeration of polynomial permutation classes. arXiv:1308.4946 [math.CO].

[18] HUCZYNSKA, S., AND VATTER, V. Grid classes and the Fibonacci dichotomy for restricted permutations. *Electron. J. Combin. 13* (2006), R54, 14 pp.

[19] JOHANSSON, F., AND NAKAMURA, B. Using functional equations to enumerate 1324-avoiding permutations. *Adv. in Appl. Math. 56* (2014), 20–34.

[20] KAISER, T., AND KLAZAR, M. On growth rates of closed permutation classes. *Electron. J. Combin. 9*, 2 (2003), Paper 10, 20 pp.

[21] KLAZAR, M. Bell numbers, their relatives, and algebraic differential equations. *J. Combin. Theory Ser. A 102*, 1 (2003), 63–87.

[22] KNUTH, D. E. *The Art of Computer Programming. Volume 1.* Addison-Wesley Publishing Co., Reading, Mass., 1968.

[23] KREMER, D. Permutations with forbidden subsequences and a generalized Schröder number. *Discrete Math. 218*, 1-3 (2000), 121–130.

[24] KREMER, D. Postscript: "Permutations with forbidden subsequences and a generalized Schröder number". *Discrete Math. 270*, 1-3 (2003), 333–334.

[25] KREMER, D., AND SHIU, W. C. Finite transition matrices for permutations avoiding pairs of length four patterns. *Discrete Math. 268*, 1-3 (2003), 171–183.

[26] LE, I. Wilf classes of pairs of permutations of length 4. *Electron. J. Combin. 12* (2005), Paper 25, 27 pp.

[27] NOONAN, J., AND ZEILBERGER, D. The enumeration of permutations with a prescribed number of "forbidden" patterns. *Adv. in Appl. Math. 17*, 4 (1996), 381–407.

[28] THE ON-LINE ENCYCLOPEDIA OF INTEGER SEQUENCES. Published electronically at `http://oeis.org/`.

[29] VATTER, V. Finitely labeled generating trees and restricted permutations. *J. Symbolic Comput. 41*, 5 (2006), 559–572.

[30] VATTER, V. Problems and conjectures presented at the problem session. In *Permutation Patterns*, S. Linton, N. Ruškuc, and V. Vatter, Eds., vol. 376 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 2010, pp. 339–344.

[31] VATTER, V. Small permutation classes. *Proc. Lond. Math. Soc. (3) 103* (2011), 879–921.

[32] WILF, H. S. What is an answer? *Amer. Math. Monthly 89*, 5 (1982), 289–292.

[33] ZEILBERGER, D. Enumerative and algebraic combinatorics. In *Princeton Companion to Mathematics*, T. Gowers, Ed. Princeton University Press, 2008, pp. 550–561.

APPENDIX A. VERIFICATION OF ANALYTICITY

Given the system

$$\begin{aligned}
E &= 1 + xE + xS_0 \\
S_p &= x(S_n + S_p) + \frac{x}{u}(S_p - S_0) + xD_0 \\
S_n &= E + u(S_n + S_p) + u^2(D_n + D_p) \\
D_p &= x(D_n + D_p) + \frac{x}{u}(D_p - D_0) \\
D_n &= S_n + S_p
\end{aligned}$$

Maple provides the following solution set for $\{E, S_p, S_n, D_p, D_n\}$ in terms of $\{x, u, S_0, D_0\}$:

$$E = \frac{1+xS_0}{1-x}$$

$$S_p = \frac{ux(x-1)(u^3-u^2x+u^2-u+x)D_0 - x(u^3x-u^2x^2-u^3+2u^2x+ux^2-u^2-ux+x^2+u-x)S_0 - ux(ux-u+x)}{(x-1)(u^4-3u^3x+u^2x^2+u^3-ux^2-u^2+2ux-x^2)}$$

$$S_n = \frac{u^3x(x-1)(x-u)D_0 + x(u^3x-u^2x^2-u^3+u^2x+ux^2-ux+x^2)S_0 + (ux-u+x)^2}{(x-1)(u^4-3u^3x+u^2x^2+u^3-ux^2-u^2+2ux-x^2)}$$

$$D_p = \frac{-x(x-1)(u^3+u^2-u+x)D_0 + ux^2(u-1)S_0 + xu(u-x)}{(x-1)(u^4-3u^3x+u^2x^2+u^3-ux^2-u^2+2ux-x^2)}$$

$$D_n = \frac{xu(x-1)(u^2-u+x)D_0 - x(u-1)(ux-u+x)S_0 + (x-u)(ux-u+x)}{(x-1)(u^4-3u^3x+u^2x^2+u^3-ux^2-u^2+2ux-x^2)}.$$

We need to verify that when the correct substitutions for D_0 and S_0 are made, that each of E, S_p, S_n, D_p, D_n is analytic at $(x,u) = (0,0)$. Recall that D_0 is the combinatorial root of the minimal polynomial

$$\begin{aligned}(2x^5+8x^4-x^3)D_p(x,0)^4 - (x^5+3x^4-23x^3+4x^2)D_p(x,0)^3 \\ +(2x^4-4x^3+20x^2-4x)D_p(x,0)^2 \\ -(x^3-4x^2-4x+1)D_p(x,0) \\ +x = 0\end{aligned}$$

and S_0 is the combinatorial root of the minimal polynomial

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

It remains to show that there are functions, analytic in a neighborhood of $(0,0)$ that agree with the right hand sides of the expression above whenever the denominators are not zero. In other words we must show that the denominator,

$$D(x,u) = (x-1)(u^4-3u^3x+u^2x^2+u^3-ux^2-u^2+2ux-x^2)$$

is a factor of each of the numerators in the ring $\mathbb{C}\langle x,u\rangle$ the ring of formal power series in x and u having complex coefficients that converge in some neighborhood of $(0,0)$.

The following facts are specializations of more general ones which can be found in, e.g., [14, Section 6.13]:

- $\mathbb{C}\langle x,u\rangle$ is a unique factorization domain.
- If $f, g \in \mathbb{C}\langle x,u\rangle$ with f irreducible, and if in some neighborhood of $(0,0)$ $f(x,u) = 0$ implies $g(x,u) = 0$, then f is a divisor of g.
- If $f \in \mathbb{C}\langle x,u\rangle$ is actually a monic polynomial in x over $\mathbb{C}\langle u\rangle$, then it is irreducible in $\mathbb{C}\langle x,u\rangle$ if and only if it is irreducible as a polynomial over $\mathbb{C}\langle u\rangle$.

The solution for $E(x)$ is easily seen to be analytic at the origin, so we focus on the remaining four functions. The denominator $D(x,u)$ is zero along the curves

$$x_1(u) = \frac{u(3u^2 - 2 + \sqrt{5}u^2)}{2(u^2 - u - 1)}, \qquad x_2(u) = \frac{u(3u^2 - 2 - \sqrt{5}u^2)}{2(u^2 - u - 1)}, \qquad x_3(u) = 1$$

in $\mathbb{C}^2$. In fact, up to sign, the denominator factors as $(x - x_1(u))(x - x_2(u))(x - x_3(u))$. Each of these factors is irreducible in $\mathbb{C}\langle x,u\rangle$ as they are linear polynomials in x over $\mathbb{C}\langle u\rangle$. The curve $x_3(u) = 1$ is of no interest to us as it does not pass through the origin. Therefore, to establish that we have analytic solutions for S_p, S_n, D_p and D_n it suffices to show that each numerator vanishes on the curves $x = x_1(u)$ and $x = x_2(u)$. Unfortunately, Maple is unable to directly verify that the numerator is zero when the correct values of S_0 and D_0 are substituted into each numerator in addition to one of the substitutions $x = x_1(u)$ or $x = x_2(u)$.[4]

Instead, we use resultant methods to first find a minimal polynomial for each numerator.[5]

The resultant of two polynomials $P(x)$ and $Q(x)$ with respect to x (denoted $\mathrm{Res}(P(x),Q(x),x)$) is a polynomial that is equal to zero if and only if $P(x)$ and $Q(x)$ have a common root. Suppose α is a root of $P(x)$ with $\deg_x(P) = d$ and β is a root of $Q(x)$ (where P, Q, α, and β may involve other variables). Then:

- $\alpha + \beta$ is a root of the resultant of $P(x - t)$ and $Q(t)$ with respect to t,
- $\alpha\beta$ is a root of the resultant of $t^d P(x/t)$ and $Q(t)$ with respect to t, and
- α/β is a root of the resultant of $P(xt)$ and $Q(t)$ with respect to t.

We now describe in detail how to find a minimal polynomial for the numerator of the solution for D_n, which we denote by $N(x,u)$. The process is the same for the other three solutions. Note that

$$N(x,u) = p_1(x,u)D_0 + p_2(x,u)S_0 + p_3(x,u),$$

where

$$\begin{aligned} p_1(x,u) &= xu\,(x-1)\left(u^2 - u + x\right), \\ p_2(x,u) &= -x\,(u-1)\,(ux - u + x)\,, \text{ and} \\ p_3(x,u) &= (x-u)\,(ux - u + x)\,. \end{aligned}$$

Let

$$\begin{aligned} P_1(P,x,u) &= P - p_1(x,u), \\ P_2(P,x,u) &= P - p_2(x,u), \text{ and} \\ P_3(P,x,u) &= P - p_3(x,u), \end{aligned}$$

so that P_1 is the minimal polynomial for the coefficient of D_0, P_2 is the minimal polynomial for the coefficient of S_0, and P_3 is the minimal polynomial for the term with neither S_0 nor D_0. Define $Q_1(D_0,x)$ and $Q_2(S_0,x)$ to be the minimal polynomials for D_0 and S_0, so that

$$Q_1(D_0,x) = (2x^5 + 8x^4 - x^3)D_0^4 - (x^5 + 3x^4 - 23x^3 + 4x^2)D_0^3$$

[4] Of course, one can compute series expansions to any degree desired, but this does not constitute a proof.

[5] We thank David Bevan for making us aware of these methods.

$$+(2x^4-4x^3+20x^2-4x)D_0^2-(x^3-4x^2-4x+1)D_0+x$$

and

$$\begin{aligned}Q_2(S_0,x) = {} & (2x^3+8x^2-x)S_0^4-(x^4+3x^3-58x^2+19x-1)S_0^3\\ & +(3x^4-30x^3+130x^2-56x+7)S_0^2-(x^4+3x^3-58x^2+19x-1)S_0\\ & +(2x^3+8x^2-x).\end{aligned}$$

Therefore, the minimal polynomial of $p_1(x,u)D_0$ is one of the irreducible factors of

$$\mathrm{Res}(tP_1(P/t,x,u),Q_1(t,x),t)$$

and the minimal polynomial of $p_2(x,u)S_0$ is one of the irreducible factors of

$$\mathrm{Res}(tP_2(P/t,x,u),Q_2(t,x),t).$$

In each of these cases, only one of the irreducible factors involves x and u, and so these must be the minimal polynomials. Call them $R_1(P,x,u)$ and $R_2(P,x,u)$ respectively.

The minimal polynomial of $p_1(x,u)D_0+p_2(x,u)S_0$ is thus one of the irreducible factors of

$$T_1(P,x,u)=\mathrm{Res}(R_1(P-t,x,u),R_2(t,x,u),t).$$

It turns out that $T_1(P,x,u)$ has four irreducible factors:

$$T_1(P,x,u)=T_{1a}(x)T_{1b}(P,x,u)T_{1c}(P,x,u)T_{1d}(P,x,u).$$

Clearly, $T_{1a}(x)$ is not the minimal polynomial that we seek. Moreover, by taking power series expansions, we see that the other three factors are minimal polynomials for:

$$\begin{aligned}T_{1b}(P,x,u)=0 &\implies P=(u^2-u)-(5u+1)(u-1)x+\cdots\\ T_{1c}(P,x,u)=0 &\implies P=-\frac{7-3\sqrt{5}}{2}(u^2-u)x+\cdots\\ T_{1d}(P,x,u)=0 &\implies P=-(u^3-2u^2+u)x^2+\cdots\end{aligned}$$

and so by matching with the known expansion for $p_1(x,u)D_0+p_2(x,u)S_0$, we set

$$\widehat{T}_1(P,x,u)=T_{1d}(P,x,u).$$

Finally, the minimal polynomial of the entire numerator is an irreducible factor of

$$T_2(P,x,u)=\mathrm{Res}(\widehat{T}_1(P-t,x,u),P_3(t,x,u),t)$$

and by another factor check we find that the minimal polynomial for the numerator, which we call $\widehat{T}_2(P,x,u)$.[6]

At this stage, we use Maple to verify that substituting either $x=x_1(u)$ or $x=x_2(u)$ into the appropriate root of $\widehat{T}_2$ yields zero. Maple code for this procedure can be found as an ancillary file at

http://arxiv.org/src/1510.00269/anc/MapleVerification.txt.

It follows that the denominator $D(x,u)$ of the four bivariate solutions divides the numerator as we required.

[6] In this instance the resultant is actually already irreducible so there is nothing to check.