Combinatorial Exploration: An Algorithmic Framework For Enumeration

Michael H. Albert
Department of Computer Science
University of Otago
Dunedin, New Zealand
malbert@cs.otago.ac.nz

Christian Bean
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
christianbean@ru.is

Anders Claesson
Division of Mathematics
The Science Institute
University of Iceland
Reykjavik, Iceland
akc@hi.is

Émile Nadeau
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
emile19@ru.is

Jay Pantone
Department of Mathematical
and Statistical Sciences
Marquette University
Milwaukee, WI, USA
jay.pantone@marquette.edu

Henning Ulfarsson
Department of Computer Science
Reykjavik University
Reykjavik, Iceland
henningu@ru.is

Combinatorial Exploration is a new domain-agnostic algorithmic framework to automatically and rigorously study the structure of combinatorial objects and derive their counting sequences and generating functions. We describe how it works and provide an open-source Python implementation. As a prerequisite, we build up a new theoretical foundation for combinatorial decomposition strategies and combinatorial specifications.

We then apply Combinatorial Exploration to the domain of permutation patterns, to great effect. We rederive hundreds of results in the literature in a uniform manner and prove many new ones. These results can be found in a new public database, the Permutation Pattern Avoidance Library (PermPAL) at https://permpal.com. Finally, we give three additional proofs-of-concept, showing examples of how Combinatorial Exploration can prove results in the domains of alternating sign matrices, polyominoes, and set partitions.

1. Introduction

Combinatorial structures are ubiquitous throughout mathematics. Graphs, permutations, words, polyominoes and other families of combinatorial objects often play a central role in many different fields. Enumerative combinatorics is concerned with the elucidation of structural properties of these families such as counting, classification, and limiting behavior.

A *combinatorial set* is a set of combinatorial objects, each of which is assigned a nonnegative integer size, with the property that the number of objects of any given size is finite. The analysis of a given combinatorial set often relies on domain-specific methods and ad-hoc derivations. In this work, we develop an algorithmic framework capable of automatically deriving rigorous proofs about the structure of combinatorial sets and computing their counting sequences and generating functions.

At its heart, *Combinatorial Exploration* is the systematic application of structural strategies that break down a set of combinatorial objects repeatedly into simpler parts, until a full decomposition into completely understood parts is obtained. The quickly growing fields of *symbolic combinatorics* and *analytic combinatorics* revolve around techniques to derive enumerative information when the structure of a combinatorial set is already understood. The literature already contains several effective methods to describe the structure of a combinatorial set and the most fundamental is the theory of combinatorial species, developed by Joyal [97, 98] in the 1980s and put into book form by Bergeron, Labelle, and Leroux [24] in the 1990s. More recently, Pivoteau, Salvy, and Soria [123] have studied recursively defined species from a computational perspective. The constructible classes of Flajolet and Sedgewick [81] can be seen as a more narrowly focused form of combinatorial species that avoids the language of category theory. Another device for describing the construction of a combinatorial set is the generating tree of Chung, Graham, Hoggatt, and Kleiman [63]; it was further formalized in the ECO method of Barcucci, Lungo, Pergola, and Pinzani [19]. The starting point of all of these approaches is foreknowledge of the structure of a combinatorial set. Combinatorial Exploration addresses the absence of a uniform method for completing that first step: understanding the structure.

This article is accompanied by an implementation of Combinatorial Exploration that can be used to apply the algorithms presented here to specific enumerative problems. That implementation consists of around 7,300 lines of Python code and is available on Github [22]. Under our "plug-and-play" framework, designed specifically to enable researchers to use Combinatorial Exploration on new domains, one needs only to provide an implementation of their specific combinatorial sets together with whichever domain-specific structural strategies they wish to use. In this way, we believe that Combinatorial Exploration constitutes the first component of a new toolkit for enumerative combinatorics. The Github repository [22] provides a short tutorial to implement Combinatorial Exploration for the easy domain of binary words using only about 200 lines of code.

To prove the efficacy of this new framework, we have also applied Combinatorial Exploration to the domain of permutation patterns. The result is that we are able to automatically and rigorously prove the results of dozens of existing papers as well as many new ones. That implementation is also available on Github [21], and consists of around 24,000 lines of Python code. We have additionally created an online database presenting the results of this work in the field of permutation patterns. It is called the Permutation Pattern Avoidance Library (PermPAL) [6], and can be found at `https://permpal.com`. It is inspired by Tenner's Database of Permutation Pattern Avoidance [131], the `graphclasses.org` website [124], and of course the Online Encyclopedia of Integer Sequences [128].

Section 2 gives a brief introduction to the field of permutation patterns, which will be used as an example domain throughout the paper. In particular, Subsection 2.4 catalogs many of the successful applications of Combinatorial Exploration in this domain.

Section 3 lays down a theoretical framework for concepts that have been widely used in enumerative combinatorics for some time. When Combinatorial Exploration is successful, the result is a *combinatorial specification* that fully describes the combinatorial set and whose validity is rigorously verified. By introducing a new type of decomposition function called a *combinatorial strategy*, we are able to create a more formalized version of combinatorial specifications. This section then introduces Combinatorial Exploration and its associated suite of combinatorial algorithms.

Section 4 provides results that guarantee that a combinatorial specification produced by Combinatorial Exploration is *productive*, that is, it contains sufficient information to uniquely determine the counting sequences of the involved sets. We believe that the formalizations created and explored in Sections 3 and 4 represent, independently of Combinatorial Exploration, important advances in the understanding of combinatorial specifications and their use to solve enumerative problems.

Section 5 describes the enumerative and analytic tools that derive various enumerative products from a combinatorial specification, including polynomial-time counting algorithms, generating functions, and random sampling procedures.

Sections 6, 7, 8, and 9 demonstrate the impressive efficacy and wide applicability of Combinatorial Exploration by applying it to four domains: permutation patterns, alternating sign matrices, polyominoes, and set partitions, respectively. The domain of permutation patterns is afforded the most attention and presented in the most detail, and this is the domain that we have actually implemented [21]. For each of the remaining three domains, we only briefly describe one possible way to approach them with Combinatorial Exploration, and derive by hand one or two combinatorial specifications for interesting combinatorial sets. For alternating sign matrices, we find a specification for the set of 132-avoiding alternating sign matrices, which has an algebraic generating function. For polyominoes, we present a specification that enumerates Ferrers diagrams and briefly discuss one for moon (L-convex) polyominoes, both of which have non-D-finite generating functions. For set partitions, we derive specifications for both the 1212-avoiding and the 111-avoiding set partitions; the former has an algebraic generating function, while the latter has a D-finite generating function. We expect that Combinatorial Exploration would be equally successful in many other domains, such as inversion sequences, ascent sequences, and more.

Finally, Section 10 discusses some algorithmic aspects of our implementation of Combinatorial Explorations, and Section 11 provides a few concluding remarks and suggests avenues for continued development.

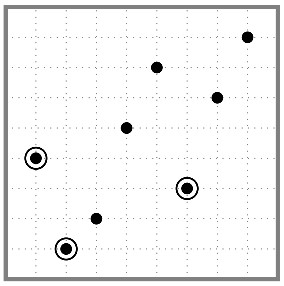


Figure 1: The permutation 41257368 with a circled occurrence of the pattern 312.

2. Permutation Patterns

Although Combinatorial Exploration is inherently a domain-agnostic tool, it will prove useful in the following sections to have an interesting domain at hand. To that end, this section aims to be a crash course on the field of *permutation patterns*, an active area of enumerative combinatorics for several decades that has meaningful connections to computer science, statistical mechanics, and algebraic geometry. In this section we give a brief introduction to the field and provide a summary of the many cases in which Combinatorial Exploration rigorously proves both new and old results.

2.1 Introduction to Permutation Patterns

A *permutation* of length n is an ordering of the numbers $1, 2, \ldots, n$. The *standardization* of an ordering of n distinct positive integers is the permutation of the same length whose entries have the same relative order. For example, the standardization of 6283 is 3142, since 2 is the smallest entry in 6238 and so it becomes 1, 3 is the second smallest entry and so it becomes 2, and so on.

We say that a larger permutation $\pi = \pi(1)\pi(2)\cdots\pi(n)$ *contains* a smaller permutation $\sigma = \sigma(1)\sigma(2)\cdots\sigma(k)$ *as a pattern*, and write $\sigma \leqslant \pi$, if π has a (not necessarily consecutive) subsequence $\pi(i_1)\pi(i_2)\cdots\pi(i_k)$ whose standardization is equal to σ. In this context σ is called a *pattern* and the subsequence $\pi(i_1)\pi(i_2)\cdots\pi(i_k)$ is an *occurrence* of the pattern in π. For instance, $312 \leqslant 41257368$ as exhibited by the subsequence 413 (among others), but one can check that $321 \nleqslant 41257368$. In this case we say that 41257368 *avoids* 321.

A geometric viewpoint turns out to be useful. We can associate each permutation π of length n with the plot of the points $\{(i, \pi(i)) : 1 \leqslant i \leqslant n\}$ in the Cartesian plane. Conversely, any finite set of points in the plane such that no two points lie on the same horizontal or vertical line can be continuously deformed to a plot of this kind and associated with the resulting permutation. The permutation containment relation has a tidy geometric definition: π contains σ if some finite subset of the points in the plot for π can be deleted to leave (up to continuous deformation) the plot of σ. Figure 1 shows the plot of 41257368, and demonstrates graphically that 41257368 contains the permutation 312.

The notion of one permutation containing another is the engine that drives the entire field of permutation patterns. A *permutation class* (often abbreviated as simply a *class*) is a downward-closed set of permutations under the containment order; that is, a set $\mathcal{C}$ of permutations is a class if it has the property that if $\pi \in \mathcal{C}$ and $\sigma \leqslant \pi$, then $\sigma \in \mathcal{C}$. Every class can be uniquely

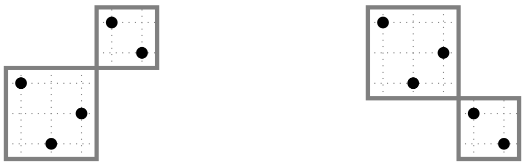


Figure 2: On the left, the permutation $312 \oplus 21 = 31254$, and on the right, the permutation $312 \ominus 21 = 53421$.

specified by the minimal set of permutations that it avoids, called its *basis*, and the class with basis B is denoted $\text{Av}(B)$. Put plainly, $\text{Av}(B)$ is the downward-closed set of permutations that avoid all of the permutations in B.

In an exercise in Chapter 2 of his seminal work *The Art of Computer Science* [104, Exercise 2.2.1.5], Knuth asks the reader to prove that the set of permutations that can be sorted by a stack is precisely $\text{Av}(231)$[1] and to find the number of permutations of each length in this set. This innocently simple exercise set off a rapid and deep exploration of permutation classes. We recommend Vatter's survey [137] for a comprehensive introduction to the field.

2.2 Sums and Skew Sums of Permutations

We will rely on two particular operations on permutations to give illustrative examples in the following sections, and we introduce those now. Given permutations σ and τ of lengths k and ℓ respectively, the *(direct) sum* $\sigma \oplus \tau$ is the permutation of length $k + \ell$ defined by

$$(\sigma \oplus \tau)(i) = \begin{cases} \sigma(i), & 1 \leqslant i \leqslant k \\ \tau(i-k) + k, & k+1 \leqslant i \leqslant k+\ell \end{cases}.$$

Geometrically, $\sigma \oplus \tau$ is formed by placing the entries of τ above and to the right of the entries of σ, as demonstrated in Figure 2.

Symmetrically, the skew sum of σ and τ, denoted $\sigma \ominus \tau$ is formed geometrically but placing the entries of τ below and to the right of the entries of σ. Formally,

$$(\sigma \ominus \tau)(i) = \begin{cases} \sigma(i) + \ell, & 1 \leqslant i \leqslant k \\ \tau(i-k), & k+1 \leqslant i \leqslant k+\ell \end{cases}.$$

These two operations can be extended to sets in a natural way. For any two sets of permutations $\mathcal{A}$ and $\mathcal{B}$, we define

$$\mathcal{A} \oplus \mathcal{B} = \{\alpha \oplus \beta \ : \ \alpha \in \mathcal{A}, \beta \in \mathcal{B}\}$$

and

$$\mathcal{A} \ominus \mathcal{B} = \{\alpha \ominus \beta \ : \ \alpha \in \mathcal{A}, \beta \in \mathcal{B}\}.$$

2.3 Enumeration of Permutation Classes

Letting $\mathcal{C}_n$ denote the set of permutations of length n in $\mathcal{C}$, we define the *counting sequence* of a class to be the sequence $a_n = |\mathcal{C}_n|$, with the convention that $\mathcal{C}_0$ contains the empty permutation

[1]We omit the set braces in this notation, writing e.g., $\text{Av}(123, 3412)$ instead of $\text{Av}(\{123, 3412\})$.

of length 0 and thus has cardinality 1. The *generating function* of $\mathcal{C}$ is the formal power series for its counting sequence:

$$f_{\mathcal{C}}(x) = \sum_{n \geqslant 0} |\mathcal{C}_n| x^n.$$

A *principal class* is one whose basis contains only a single permutation. The simplest nontrivial examples are $\text{Av}(12)$ and $\text{Av}(21)$. The former contains all decreasing permutations (e.g., 87654321), while the latter contains all increasing permutations. The counting sequences of these two classes are clearly the same, with $a_n = 1$ for all n, and their generating function $f_{\text{Av}(12)} = f_{\text{Av}(21)} = 1/(1-x)$ is rational.

Principal classes of length 3. The six principal classes avoiding a pattern of length 3 are $\text{Av}(123)$, $\text{Av}(132)$, $\text{Av}(213)$, $\text{Av}(231)$, $\text{Av}(312)$, and $\text{Av}(321)$. The outer two are essentially the same[2] when accounting for symmetries of the permutation containment order, and so are the middle four. Although these two groupings, called *symmetry classes*, are structurally very different[3], it turns out that they are both counted by the famous Catalan numbers $a_n = \binom{2n}{n}/(n+1)$ and have the algebraic generating function $(1-\sqrt{1-4x})/(2x)$. We will be making frequent references to the Online Encyclopedia of Integer Sequences [128] (OEIS), which hosts integer sequences and vast information about them, e.g., the Catalan numbers appear as sequence A000108 on the site.

Principal classes of length 4. We now arrive at the twenty four principal classes avoiding a pattern of length 4, and already we encounter the active edge of investigation. These twenty four principal classes partition into seven symmetry classes. Two turn out to have the same algebraic generating function (see Bóna [38] and Stankova [130]) and four turn out to have the same D-finite generating function[4] (see Bousquet-Mélou [42], Gessel [85], Stankova [129], and West [138]). The generating function for the last symmetry class, containing $\text{Av}(1324)$ and $\text{Av}(4231)$, remains unknown. Through about a dozen articles [8, 25, 28, 29, 36, 37, 64, 66, 67, 95, 117], the first fifty terms of the counting sequence of $\text{Av}(1324)$ are known explicitly and rough bounds on the exponential growth are known.

The 2 × 4 classes. For the last 25 years, a particular collection of permutation classes has been perhaps the most important and influential in guiding the direction of research. A *2×4 class* is a permutation class whose basis consists of two permutations of length four, e.g., $\text{Av}(2413, 3142)$. Accounting for symmetries of the pattern containment relation, there are essentially 56 different 2×4 classes. The quest to understand these 56 permutation classes has led to the creation of a surprising number of sophisticated enumerative techniques that we mention below.

Automatic methods in permutation patterns. Automatic and computational methods, both rigorous and empirical, have played a prominent role in the study of permutation patterns for quite some time. Figure 3 shows many of these methods. An arrow from method A to method

[2]To be more precise, there are obvious bijections between them, derived from the symmetries of the square, that preserve both length and permutation containment.

[3]To mention just one difference, $\text{Av}(123)$ contains infinite antichains under the containment order, while $\text{Av}(132)$ does not.

[4]A generating function is *D-finite* if it is the solution of a nontrivial linear differential equation with polynomial coefficients.

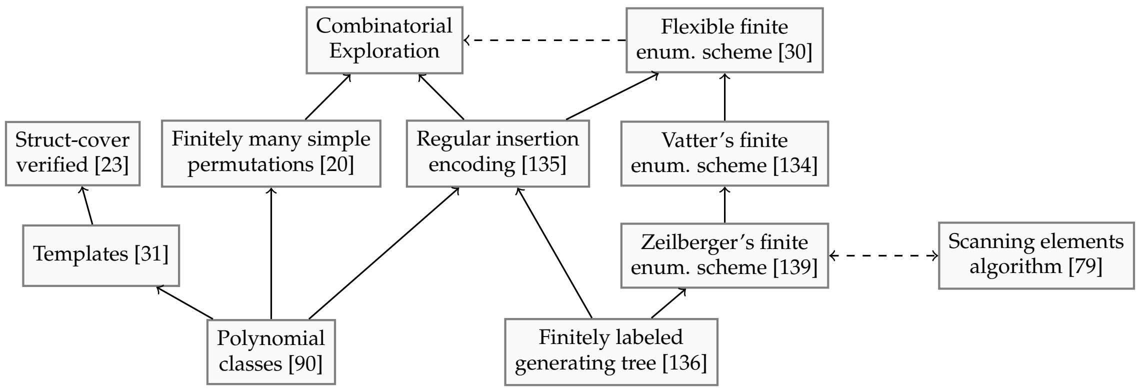

Combinatorial Exploration
Flexible finite enum. scheme [30]
Struct-cover verified [23]
Finitely many simple permutations [20]
Regular insertion encoding [135]
Vatter's finite enum. scheme [134]
Templates [31]
Zeilberger's finite enum. scheme [139]
Scanning elements algorithm [79]
Polynomial classes [90]
Finitely labeled generating tree [136]

Figure 3: Comparison of algorithmic enumeration methods.

B implies that method B is stronger, i.e., every permutation class that can be enumerated automatically by method A can also be enumerated by method B. A dashed arrow implies that we conjecture that method B is stronger than method A. As the figure shows, several of these methods, e.g., Zeilberger's enumeration schemes, Vatter's implementation of the regular insertion encoding, and the substitution decomposition applied to permutation classes with finitely many simples, are subsumed by Combinatorial Exploration.

2.4 Success of Combinatorial Exploration in Permutation Patterns

Before introducing Combinatorial Exploration over the next several sections, we hope to first convince the reader that it is a powerful enumerative tool by describing some of the new and old results that it can automatically and rigorously prove. We strongly believe that no algorithmic effort can fully substitute for the endless creativity of human mathematicians. There are, however, many examples where the enumeration of a specific combinatorial set is needed as a stepping stone on the way to a more important theorem, and for tasks like this Combinatorial Exploration can be tremendously helpful.

As Section 3 describes, when the process of Combinatorial Exploration is successful, the result is a rigorous description of the permutation class called a *combinatorial specification*, and as Section 5 discusses, this combinatorial specification can be converted to a system of equations satisfied by the generating function for the class.

In many cases, this is a univariate algebraic system—each equation expresses one of the generating functions as the sum or product of others—and can be solved explicitly or implicitly with an appropriate computer algebra system[5]. In other cases, we get a bivariate algebraic system with one catalytic variable (to use the terminology of Zeilberger [141]). These can be solved by adapting the methods of Bousquet-Mélou and Jehanne [44] from single equations to systems of equations, although the symbolic computations required (in particular, elimination via Gröbner bases or resultants) can be intense. These systems generically have solutions that are algebraic.

Lastly, in some cases we obtain systems of equations with two or more catalytic variables, and although we know of no way to solve these systems, they nonetheless permit calculation of

[5]We find Maple particularly effective.

the terms of the counting sequences involved in polynomial time. It is known that solutions of these systems can be non-D-finite. In the remainder of this section, we use “CV” as an abbreviation for “catalytic variable”.

Let us finally say, most importantly, that the successes of Combinatorial Exploration that are reproved results from the literature are not obtained by simply taking the ideas from that literature and implementing them as code. Instead, as described in Section 6, we have come up with just a small handful of what we call *combinatorial strategies* that are sufficient to yield all of the results below in a surprisingly uniform manner, automatically and rigorously.

- ⋄ We can find specifications automatically for six out of the seven symmetry classes of permutations avoiding one pattern of length 4, all but Av(1324). Four of these six, Av(1234), Av(1243), Av(1432), and Av(2143), have the same counting sequence, and we find for them specifications with two CVs, which is consistent with their shared generating function being known to be non-algebraic. Additionally, the first three of these have specifications that are *parallel*, which means they have a similar enough structure that one can automatically produce bijections between any pair of them. The remaining two of the six, Av(1342) and Av(2413), also share a counting sequence, and we find specifications with one CV that can easily be solved to produce their algebraic generating function. These are the first *direct* enumerations of these two permutation classes, as previous enumerations were via bijections to each other and to other objects. The final class, Av(1324), currently remains out of reach, but we are optimistic that several not-yet-implemented strategies may lead to progress.
- ⋄ Out of the 56 symmetry classes of permutations avoiding two patterns of length 4—the classes that have been an important testbed for new enumerative techniques over the years [2–5, 9, 16, 18, 26, 27, 35, 39, 49, 50, 105–107, 109, 118, 120, 122, 135] —we can find specifications for all 56 of them. 53 have specifications with 0 CVs or 1 CV, and so their algebraic generating functions can be determined. The remaining three were conjectured by Albert, Homberger, Pantone, Shar, and Vatter [9] to be non-D-finite, and correspondingly their specifications have 2 or more CVs. Figure 9 on page 37 shows heatmaps for the 2×4 classes, derived by sampling many long permutations uniformly at random from each class.
- ⋄ Out of the 317 symmetry classes of permutations avoiding three patterns of length 4, again we can find specifications for all of them. One is conjectured to be non-D-finite, and for this class we find a specification with 2 CVs. For the remaining 316 we find specifications with 0 CVs or 1 CV. The original enumeration of these classes required an enormous amount of work, largely by hand, and can be found in [9, 53–59].
- ⋄ Similarly, we can find specifications for all symmetry classes avoiding n patterns of length 4 for $4 \leqslant n \leqslant 24$, all with 0 CVs or 1 CV, implying that we can compute all of their generating functions. This work was originally done by Mansour with various collaborators [51, 52, 112–116], requiring hundreds of pages of work, and even then with a vast majority of the derivations left as exercises for the reader.
- ⋄ As Figure 3 explained, Combinatorial Exploration can automatically enumerate any insertion encodable permutation class. More significantly, it can do so much more quickly than Vatter’s original implementation, which requires a great deal of brute force genera-

tion of permutations to determine which configurations are possible and which are not. The particular data structure we use in Combinatorial Exploration for permutation patterns (described in Section 6) can automatically determine this information without such generation. As a result, we can enumerate insertion encodable classes that were previously out of reach due to computational intensity.

- ⋄ Bevan, Brignall, Elvey Price, and Pantone [28] found improved lower and upper bounds on the exponential growth rate of Av(1324) by considering a set of gridded permutations that they called "domino permutations". The enumeration of these was challenging, requiring a bijection to a type of arch systems and several pages of work to enumerate these arch systems. We can find a specification and the algebraic generating function for the domino permutations.

- ⋄ Ikeda's thesis [91] focuses on the enumeration of Av(52341, 53241, 52431, 35142, 42513, 351624), which consist of permutations representing the *local complete intersection* Schubert varieties. These Schubert varieties have singularities that are well-behaved, in the sense that their local rings satisfy a certain algebraic condition, see e.g., Ulfarsson and Woo [133]. The main result in Ikeda's thesis, Theorem 6.9, is incorrect. We are able to find a specification with 1 CV and the correct algebraic generating function, which satisfies the equation

$$\begin{aligned}0 = &(13x^9 - 69x^8 + 162x^7 - 115x^6 - 496x^5 + 1401x^4 - 1415x^3 + 637x^2 - 97x + 4)F(x)^3 \\ &+ (50x^8 - 198x^7 + 376x^6 + 118x^5 - 1794x^4 + 2840x^3 - 1676x^2 + 281x - 12)F(x)^2 \\ &+ (60x^7 - 192x^6 + 248x^5 + 483x^4 - 1743x^3 + 1472x^2 - 272x + 12)F(x) \\ &+ 24x^6 - 64x^5 + 40x^4 + 295x^3 - 432x^2 + 88x - 4\end{aligned}$$

- ⋄ Defant [69] studies the preimage of various permutation classes under the West-stack-sorting operation, derives that the preimage of Av(321) is Av(34251, 35241, 45231), and gives rough bounds on its exponential growth rate, but is unable to enumerate it. We find a specification with 2 CVs, allowing us to compute 636 terms in the counting sequence. We are unable to conjecture the generating function from these terms, and thus we predict that it is non-D-finite. Empirically, we estimate with quite a bit of confidence that the growth rate is $6 + 2\sqrt{5}$.

- ⋄ Bóna and Pantone [40] used Combinatorial Exploration to assist with the study of five classes avoiding four patterns of length 5, and one class avoiding five patterns of length 6. All have specifications with 2 or more CVs, and they were able to predict that several had D-finite generating functions, and give empirical estimates of their asymptotic behavior.

- ⋄ Dimitrov [71] studies "box classes". An incorrect theorem for the enumeration of Av($1 \square 3 \square 2$) is given [71, Theorem 4.4]. We are able to find a specification with 1 CV and compute the correct algebraic generating function, which satisfies the equation

$$\begin{aligned}0 = &x^8F(x)^6 - x^2(x^6 + 2x^5 + 5x^4 + 2x^3 + 4x^2 - 4x + 5)F(x)^4 \\ &+ (2x^4 + 2x^3 + 4x^2 - 2x + 1)F(x)^2 - 1\end{aligned}$$

  We can also enumerate many other box classes, including Av($1 \square 2 \square 3$), Av($1 \square \square 32$), and Av($13 \square \square 2$).

- ⋄ Egge [72] conjectured that a group of permutation classes defined by avoiding two patterns of length 4 and one of length 6 are all counted by the Schröder numbers. Burstein and Pantone [48] proved one of these conjectures, and then Bloom and Burstein [32] proved the remainder. We are able to find specifications and generating functions for all of these classes.
- ⋄ The skew-vexillary permutations are those in the class Av(24153, 25143, 31524, 31542, 32514, 32541, 42153, 52143, 214365). They were studied by Klein, Lewis, and Morales [103] who showed that their generating function $S(x)$ can be written in terms of the generating function $V(x)$ for Av(2143) as

  $$S(x) = (1-x)V(x)^2 - V(x) - \frac{1}{1-x}.$$

  We are able to find a specification for the skew-vexillary permutations expressed in terms of the subclass Av(2143). This specification treats Av(2143) as a class whose generating function is already independently known, and it produces the same equation relating $S(x)$ and $V(x)$.
- ⋄ The juxtaposition of two classes $\mathcal{C}$ and $\mathcal{D}$ is the class of permutations that can be formed by putting a permutation from $\mathcal{C}$ side-by-side with a permutation from $\mathcal{D}$ and vertically interleaving the entries in some way. The juxtaposition of two finitely-based classes is itself a finitely-based class. We can enumerate many juxtaposition classes, including the interesting case $\mathcal{C} = \text{Av}(213, 231)$ and $\mathcal{D} = \text{Av}(132, 312)$, which is an example of two classes with rational generating functions whose juxtaposition is algebraic (and non-rational).
- ⋄ Staircase classes were considered by Albert, Pantone, and Vatter [10]. The (Av(21), Av(21))-staircase is just Av(321), but the (Av(12), Av(21))-staircase is much more interesting. It is predicted to be the class Av(2341, 3421, 4231, 52143). We find a specification with 2 CVs, and compute 400 terms of the counting sequence. We are unable to conjecture its generating function, but we estimate the growth rate to be about 4.1768325. In the same paper, the class Av(4321, 321654, 421653, 431652, 521643, 531642) is given as a potential counterexample to a possible general property—we find a 2 CV specification that permits us to generate enough terms of the counting sequence to determine that this class is unlikely to be such a counterexample.
- ⋄ Guo and Kitaev [83] explore the notion of "partially ordered permutations". We are able to find specifications for many of the classes they consider.
- ⋄ Elder [73] determined the basis of the class of permutations that can be sorted by a stack of depth 2 and an infinite stack in series; it has basis elements of length at most 8. The algebraic generating function of this class was found later by Elder, Lee, and Rechnitzer [74]. We find a specification with 1 CV, but the corresponding system of equations is too large for us to be able to solve and produce the algebraic solution.
- ⋄ The "Widdershins spirals" are a class of permutations whose generating function was given without proof by Brignall, Engen, and Vatter [45] and that plays an important role in later work of Brignall and Vatter [46]. We find a specification with 0 CVs and the corresponding rational generating function.
- ⋄ The Schubert varieties with a simple presentation for their cohomology ring are studied

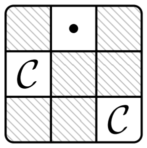


Figure 4: A graphical depiction of an arbitrary nonempty permutation in the class $\mathcal{C}$ of 132-avoiding permutations.

by Gasharov and Reiner [84] and are called varieties *defined by inclusion*. They showed that these correspond to the permutation class Av(4231, 35142, 42513, 351624). The class was enumerated in Albert and Brignall [7]. We find a specification with 0 CVs and the corresponding algebraic generating function.

◇ Alland and Richmond [12] show that for a permutation π, the Schubert variety X_π has a complete parabolic bundle structure if and only if $\pi \in$ Av(3412, 52341, 635241). We are able to find a specification with 1 CV, guaranteeing that this class has an algebraic generating functions, but the system is too large for us to solve. We can, however, use the tree to generate the first 400 terms of the counting sequence and then use these terms to conjecture a value for the generating function; it appears to be algebraic with a minimal polynomial of degree 6.

There are of course some permutation classes that appear in the literature that we are not currently able to enumerate with Combinatorial Exploration, although we are still in the process of creating new strategies that we believe have potential to help. A few examples are listed below.

◇ Burstein and Pantone [48] conjecture that Av(2143, 246135) is equinumerous to Av(2413). The generating function of the second class is already known, but we are unable to find a specification for the first class.

◇ Atkinson, Ruškuc, and Smith [17] prove that the substitution closure of the class is Av(123) is equal to the class Av(24153, 25314, 31524, 41352, 246135, 415263). We have not yet found a specification for this class.

◇ Fink, Mészáros, and St. Dizier [78] prove that the Schubert polynomial $\mathfrak{S}_\pi$ is zero-one if and only if π is in the class Av(12543, 13254, 13524, 13542, 21543, 125364, 125634, 215364, 215634, 315264, 315624, 315642). We have not found a specification for this class.

◇ The permutation class whose enumeration would be of the most interest is Av(1324), and we are unable to find a specification for it at this time.

2.5 An Enumerative Example

We end this subsection by demonstrating one way that the counting sequence and generating function for Av(132) can be calculated. Our derivation closely tracks the concept of proof trees that will be developed in Section 3.

Let $\mathcal{C}$ = Av(132). Every permutation in $\mathcal{C}$ is either empty or contains a maximum entry. Let $\pi \in \mathcal{C}$ be a nonempty permutation of length n with maximum entry $\pi(k)$. Every entry with index less than k must have value larger than every entry with index greater than k, or else π

contains a 132 pattern in which $\pi(k)$ acts as the 3. Moreover, the entries with index less than k can together form any (non-standardized) permutation that avoids 132 and the entries with index greater than k can also together form any permutation that avoids 132. More concretely, the nonempty permutations in $\operatorname{Av}(132)$ are precisely those in the set

$$(\operatorname{Av}(132) \oplus \{1\}) \ominus \operatorname{Av}(132).$$

This is shown pictorially in Figure 4.

This is a full structural description of $\operatorname{Av}(132)$ in the sense that the set of 132-avoiding permutations of length n can be completely described recursively in terms of shorter 132-avoiding permutations. From this description we can write down a functional equation satisfied by the generating function $f(x)$ that counts 132-avoiding permutations by length. The statement "Every 132-avoiding permutation is either empty or has a maximum entry with two non-interleaving 132-avoiding permutations (one on each side)" translates directly to the equation

$$f(x) = 1 + xf(x)^2.$$

Upon applying the quadratic equation and selecting the correct root, we find $f(x) = (1 - \sqrt{1-4x})/(2x)$, which is the generating function for the Catalan numbers.

3. Combinatorial Exploration

Combinatorial Exploration seeks to fully describe the structure of a combinatorial set. In this context, a *full description* is, informally, one that accomplishes the following goal.

> The set of objects of size n in a combinatorial set $\mathcal{C}$ can be completely and uniquely constructed from the set of objects of size at most $n-1$ in $\mathcal{C}$. That is to say, a set of rules is given to construct each object of size n in $\mathcal{C}$ exactly once.

The combinatorial literature has often tried to capture this informal notion using concepts like combinatorial specifications and admissible operators (see, e.g., [81], which we discuss in more detail later in this section). In order to give the most complete picture of Combinatorial Exploration and justify the correctness of its products, we are required to develop—for, to our knowledge, the first time—a full theoretical framework that rigorously captures these oft-cited concepts.

Before we begin to build this framework we introduce, again informally, *proof trees*, a combinatorial structure that represents the output of successful Combinatorial Exploration and represents the idea of a full description mentioned above.

We rely on the domain of permutation patterns heavily in this section to explain concepts and provide examples, but both Combinatorial Exploration and the theoretical framework we develop can be applied to any type of combinatorial objects that can be algorithmically constructed and manipulated.

3.1 Proof Trees

A *proof tree* for a combinatorial set $\mathcal{C}$ is a rooted tree in the graph-theoretical sense. Each vertex v represents a combinatorial set $\mathcal{D}^{(v)}$, the root representing $\mathcal{C}$ itself. Each internal vertex

v with children $u_1, \ldots, u_m$ implicitly (in a way we will make clear later) defines a structural relationship between the parent set $\mathcal{D}^{(v)}$ and the child sets $\mathcal{D}^{(u_1)}, \ldots, \mathcal{D}^{(u_m)}$.

This relationship can take many forms. In the easiest of cases, it may be that $\mathcal{D}^{(v)}$ is the disjoint union $\mathcal{D}^{(u_1)} \sqcup \cdots \sqcup \mathcal{D}^{(u_m)}$. More generally, the parent-child relationship represents some way in which $\mathcal{D}^{(v)}$ is decomposed into simpler sets $\mathcal{D}^{(u_1)}, \ldots, \mathcal{D}^{(u_m)}$, or equivalently, how the child sets $\mathcal{D}^{(u_1)}, \ldots, \mathcal{D}^{(u_m)}$ can be combined to reconstruct the parent $\mathcal{D}^{(v)}$.

Before we bring these concepts into sharper focus, let us examine the procedure by which we described and enumerated Av(132) in the previous section through this informal lens. Figure 5 shows a proof tree for $\mathcal{C} = \text{Av}(132)$. This proof tree is essentially a pictorial representation of the following structural description.

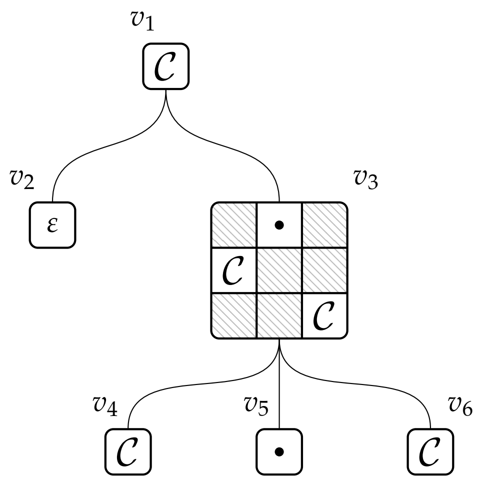


Figure 5: A proof tree for the class $\mathcal{C} = \text{Av}(132)$.

Every permutation in $\mathcal{C} = \text{Av}(132)$ is either the empty permutation of size 0 (denoted ε), or contains a topmost entry. This is represented in the proof tree by the root v_1, with $\mathcal{D}^{(v_1)} = \mathcal{C}$, having a left child v_2 representing the set consisting of only the empty permutation, and a right child v_3 representing the set of all nonempty permutations in $\mathcal{C}$. As described in the derivation of the previous section, one may observe that every entry to the left of the maximum must have greater value than every entry to the right of the maximum, and therefore all nonempty permutations are represented by the given picture. The parent-children relationship between these vertices is $\mathcal{D}^{(v_1)} = \mathcal{D}^{(v_2)} \sqcup \mathcal{D}^{(v_3)}$.

The vertex v_3 has three children, v_4, v_5 and v_6, the first and third of which represent the same set as the root. The second represents the set containing the single permutation of size 1. This relationship is *not* a disjoint union—rather, it represents that any permutation in $\mathcal{D}^{(v_3)}$ can be formed by selecting any two permutations $\alpha, \beta \in \mathcal{C}$, placing all entries in α above and to the left of all entries in β, and placing a topmost entry between the two. Moreover, every choice of α, β results in a distinct element of $\mathcal{D}^{(v_3)}$.

We have introduced the concept of proof trees specifically because they enable efficient enumeration of the objects represented by the root. We discuss this in detail in future sections, for now only showing how the proof tree above allows one to write down both a polynomial-time counting algorithm for $\mathcal{C}$ and a system of equations that can be solved to find its generating function.

Each parent-children relationship carries with it sufficient structural information to determine the number of permutations of size n in the parent from the number of permutations of various sizes in the children. When the entire proof tree is taken together, this permits the recursive computation of the number of permutations in the root of any size.

Recall that if $\mathcal{D}$ is a combinatorial set, then $\mathcal{D}_n$ denotes the set of objects in $\mathcal{D}$ with size n. As we have observed that $\mathcal{D}^{(v_1)} = \mathcal{D}^{(v_2)} \sqcup \mathcal{D}^{(v_3)}$ it clearly follows that $|\mathcal{D}_n^{(v_1)}| = |\mathcal{D}_n^{(v_2)}| + |\mathcal{D}_n^{(v_3)}|$. Furthermore, the relationship between v_3 and its children implies the equality

$$|\mathcal{D}_n^{(v_3)}| = \sum_{i=0}^{n-1} |\mathcal{D}_i^{(v_4)}||\mathcal{D}_{n-1-i}^{(v_6)}|.$$

For legibility we use the fact that $|\mathcal{D}_n^{(v_5)}| = 1$ for $n = 1$, and 0 otherwise, to avoid writing a double sum.

Noting that $\mathcal{D}^{(v_2)}$ contains only the empty permutation of size 0, we can derive from the proof tree for Av(132) the system of recurrences

$$\begin{aligned}
|\mathcal{D}_n^{(v_1)}| &= |\mathcal{D}_n^{(v_2)}| + |\mathcal{D}_n^{(v_3)}| \\
|\mathcal{D}_n^{(v_2)}| &= \begin{cases} 1, & \text{if } n = 0 \\ 0, & \text{otherwise} \end{cases} \\
|\mathcal{D}_n^{(v_3)}| &= \sum_{i=0}^{n-1} |\mathcal{D}_i^{(v_4)}||\mathcal{D}_{n-1-i}^{(v_6)}| \\
|\mathcal{D}_n^{(v_4)}| &= |\mathcal{D}_n^{(v_1)}| \\
|\mathcal{D}_n^{(v_6)}| &= |\mathcal{D}_n^{(v_1)}|
\end{aligned}$$

To derive the generating function of the class we associate to each vertex v the generating function $F_v(x) = \sum_{n \geqslant 0} |\mathcal{D}_n^{(v)}| x^n$. The structure described above allows one to write down the system of equations

$$\begin{aligned}
F_{v_1}(x) &= F_{v_2}(x) + F_{v_3}(x) \\
F_{v_2}(x) &= 1 \\
F_{v_3}(x) &= xF_{v_4}(x)F_{v_6}(x) \\
F_{v_4}(x) &= F_{v_1}(x) \\
F_{v_6}(x) &= F_{v_1}(x)
\end{aligned}$$

which can be solved to find that the generating function for Av(132) is $F_{v_1}(x) = (1 - \sqrt{1-4x})/(2x)$, as expected. We choose to return now to the viewpoint of recurrences, and we say nothing more about generating functions until Section 5.

The reader who wishes to see a larger example of a proof tree right now can refer to Figure 24 on page 57. The combinatorial sets in this proof tree for $\mathrm{Av}(1243, 1342, 2143)$ are represented by a structure we call *tilings*, introduced in Section 6.

3.2 Combinatorial Strategies

The time has finally arrived to lay the foundation for the theoretical framework upon which Combinatorial Exploration operates.

Each parent-children relationship in the proof tree for $\mathrm{Av}(132)$ in Figure 5 represents a structural decomposition of the parent combinatorial set into the child combinatorial sets. In this section we formally define this concept under the name *combinatorial strategy*.

An m-ary *combinatorial strategy* consists of three components, which we now discuss in broad terms before stating the full formal definition. The first and most important component of a strategy S is a *decomposition function* d_S, which takes as input a combinatorial set $\mathcal{A}$. If strategy S cannot be meaningfully applied to $\mathcal{A}$ to decompose it into other sets, then the output of d_S is the symbol DNA, short for "does not apply". Otherwise, the output is an m-tuple of combinatorial sets $(\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)})$.

The other two components that comprise a combinatorial strategy capture the requirement that if $d_S(\mathcal{A}) = (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)})$, then it must be possible to calculate, in a manner uniform over all input sets $\mathcal{A}$, the enumeration of $\mathcal{A}$ from the enumerations of the sets $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$. The *reliance profile function* $r_S : \mathbb{N} \to \mathbb{Z}^m$ for a strategy S encodes which terms in the enumeration of each $\mathcal{B}^{(i)}$ are necessary to calculate $|\mathcal{A}_n|$. Given an input $n \in \mathbb{N}$, $r_S(n) = (r^{(1)}(n), \ldots, r^{(m)}(n))$, where each $r^{(i)}(n) \in \mathbb{Z}$ is an upper bound for the largest size of elements in $\mathcal{B}^{(i)}$ that are necessary in order to compute $|\mathcal{A}_n|$. A negative value indicates that no values $\mathcal{B}^{(i)}$ are needed. For instance, if $d_S(\mathcal{A}) = (\mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \mathcal{B}^{(3)})$ and $r_S(10) = (2, 9, -1)$, this means that $|\mathcal{A}_{10}|$ can be computed from the quantities $|\mathcal{B}^{(1)}_0|, |\mathcal{B}^{(1)}_1|, |\mathcal{B}^{(1)}_2|$ and $|\mathcal{B}^{(2)}_0|, |\mathcal{B}^{(2)}_1|, \ldots, |\mathcal{B}^{(2)}_9|$; no values $|\mathcal{B}^{(3)}_i|$ are needed.

The third component of a strategy, the spectrum of *counting functions*, governs precisely how the enumeration of $|\mathcal{A}_n|$ is computed from the quantities $|\mathcal{B}^{(i)}_j|$. Each strategy S possesses a counting function $c_{S,(n)}$ for each $n \in \mathbb{N}$ such that when the quantities $|\mathcal{B}^{(i)}_j|$ determined by the reliance profile function are given as input to $c_{(n),S}$, then the output is $|\mathcal{A}_n|$.

We want to emphasize at this point that the reliance profile function and the counting functions of a strategy are fixed functions that do not vary with the combinatorial set $\mathcal{A}$ to which the strategy is applied. We now give the full formal definition of a strategy.

Definition 3.1. Let $\mathfrak{Z}$ be the collection of all combinatorial sets. An m-ary *combinatorial strategy* S consists of three components.

1. A *decomposition function* $d_S : \mathfrak{Z} \to \mathfrak{Z}^m \cup \{\mathtt{DNA}\}$ whose input is a combinatorial set $\mathcal{A}$ (the parent set), and whose output is either an ordered m-tuple of combinatorial sets $(\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)})$ (the child sets) or the symbol DNA. When the output is $d_S(\mathcal{A}) = \mathtt{DNA}$, short for "does not apply", we say that *S cannot be applied* to the combinatorial set $\mathcal{A}$.
2. A *reliance profile function* $r_S : \mathbb{N} \to \mathbb{Z}^m$ whose input is a natural number n and whose output is an ordered m-tuple of integers. We use $r^{(i)}_S(n)$ to denote the ith component of

$r_S(n)$, i.e.,

$$r_S(n) = (r_S^{(1)}(n), \dots, r_S^{(m)}(n)).$$

3. An infinite sequence of *counting functions* $c_{S,(n)}$ indexed by $n \in \mathbb{N}$, each of whose input is m tuples of integers $w^{(1)}, \dots, w^{(m)}$ and whose output is a natural number. The counting functions must have the property that if $d_S(\mathcal{A}) = (\mathcal{B}^{(1)}, \dots, \mathcal{B}^{(m)})$ and $r_S(n) = (r_S^{(1)}(n), \dots, r_S^{(m)}(n))$, then for input tuples

   $$w^{(i)}(n) = \left(|\mathcal{B}_0^{(i)}|, \dots, |\mathcal{B}_{r_S^{(i)}(n)}^{(i)}|\right)$$

   we have

   $$c_{S,(n)}(w^{(1)}(n), \dots, w^{(m)}(n)) = |\mathcal{A}_n|.$$

   To be overly explicit, the domain of $c_{S,(n)}$ is $\mathbb{N}^{D_1} \times \cdots \times \mathbb{N}^{D_m}$, where

   $$D_k = \max(0, r_S^{(k)}(n) + 1),$$

   while the codomain is simply $\mathbb{N}$.

The counting functions of a strategy S describe, for any combinatorial set $\mathcal{A}$ to which S can be applied and for each n separately, the uniform method of calculating $|\mathcal{A}_n|$ from the quantities $|\mathcal{B}_j^{(i)}|$ indicated by the reliance profile function. As a result, an application of a strategy $d_S(\mathcal{A}) = (\mathcal{B}^{(1)}, \dots, \mathcal{B}^{(m)})$ implies the statement:

> The number of objects in $\mathcal{A}$ of size n can be calculated from the number of objects in the $\mathcal{B}^{(i)}$ of various sizes, without regard to actual nature of the objects,

where the "various sizes" are dictated by the (fixed) reliance profile function r_S.

Many readers are no doubt familiar with the theory of symbolic combinatorics so beautifully presented by Flajolet and Sedgewick [81], to whom we owe a substantial debt of gratitude for inspiring much of the framework constructed here. In Subsection 3.3 we will say much more about the similarities and differences between their framework and ours and will discuss how what we are calling proof trees, for the moment, are simply an alternative, more pictorial, formulation of a "combinatorial specification".

Meanwhile, it is instructive at this point to give several examples and non-examples of strategies, some of which rely on notions not completely defined until Section 6, but which we hope are clear enough in context.

Example 3.1. Consider a binary strategy Z that we will call "size-0-or-not" constructed as follows.

1. The decomposition function d_Z is defined by $d_Z(\mathcal{A}) = (\mathcal{B}, \mathcal{C})$ where $\mathcal{B}$ contains precisely the elements of $\mathcal{A}$ with size 0 and $\mathcal{C}$ contains all others. It follows that $\mathcal{A}$ is the disjoint union $\mathcal{B} \sqcup \mathcal{C}$. This strategy applies to all possible input sets, and thus the output is never `DNA`.

2. The reliance profile function r_Z is $n \mapsto (n, n)$.

3. The counting functions are defined by

$$c_{Z,(n)}((b_0,\ldots,b_n),(c_0,\ldots,c_n)) = b_n + c_n$$

and since $\mathcal{A}$ is the disjoint union $\mathcal{B} \sqcup \mathcal{C}$, it is clear that the counting functions satisfy

$$c_{Z,(n)}((|\mathcal{B}_0|,\ldots,|\mathcal{B}_n|),(|\mathcal{C}_0|,\ldots,|\mathcal{C}_n|)) = |\mathcal{B}_n| + |\mathcal{C}_n| = |\mathcal{A}_n|$$

for all $n \in \mathbb{N}$, and therefore they satisfy the condition in Definition 3.1.

This is the strategy that is applied to the root of the proof tree for Av(132) in Figure 5; the output $(\mathcal{B},\mathcal{C})$ of the decomposition function gives the two children of the root.

Note that since we are guaranteed that $\mathcal{B}$ will never contain any objects of size greater than 0 we could also have defined the reliance profile function to be $n \mapsto (0,n)$, with counting functions

$$c_{Z,(n)}((b_0),(c_0,\ldots,c_n)) = \begin{cases} b_0, & n = 0 \\ c_n, & n > 0 \end{cases},$$

but we chose to highlight the disjoint union nature of this strategy.

Example 3.2. We now describe the other strategy used in the proof tree of Av(132), although we cannot give all of the details until Section 6 (here we describe just a specific case of a much more general strategy). We have also slightly simplified it by not considering the point itself to be one of the children.

Consider a binary strategy F that we will call "factor-around-max-entry" constructed as follows.

1. Let $\mathcal{A}$ be a set of permutations. If there exist sets $\mathcal{B}$ and $\mathcal{C}$ such that $\mathcal{A} = (\mathcal{B} \oplus \{1\}) \ominus \mathcal{C}$, then they are unique and we define $d_F(\mathcal{A}) = (\mathcal{B},\mathcal{C})$. Otherwise, $d_F(\mathcal{A}) = \texttt{DNA}$.

2. Since the objects in $\mathcal{A}$ of size n are built from pairs (β,γ) where $|\beta| + |\gamma| = n-1$, the reliance profile function is $n \mapsto (n-1,n-1)$. (This example demonstrates why the values in the reliance profile function come from $\mathbb{Z}$ rather than just $\mathbb{N}$.)

3. The counting functions are $c_{F,(0)} = 0$ and if $n > 0$

$$c_{F,(n)}((b_0,\ldots,b_{n-1}),(c_0,\ldots,c_{n-1}) = \sum_{j=0}^{n-1} b_j c_{n-1-j}.$$

We must show that the counting function satisfies the equality

$$|\mathcal{A}_n| = c_{F,(n)}((|\mathcal{B}_0|,\ldots,|\mathcal{B}_{n-1}|),(|\mathcal{C}_0|,\ldots,|\mathcal{C}_{n-1}|));$$

every permutation $\pi \in \mathcal{A}_n$ can be written as $(\beta \oplus 1) \ominus \gamma$ for $\beta \in \mathcal{B}$ and $\gamma \in \mathcal{C}$ in precisely one way, because β must involve the entries to the left of the maximum entry and γ must involve the entries to the right of the maximum entry.

Non-example 3.3. In the previous example, the identification of the maximum entry of a permutation as a splitting point is necessary for there to be a uniform counting function. Suppose instead that we had attempted to define a decomposition function $d_F(\mathcal{A}) = (\mathcal{B},\mathcal{C})$ if $\mathcal{A} = \mathcal{B} \oplus \mathcal{C}$, with $d_F(\mathcal{A}) = \texttt{DNA}$ otherwise. This leads to two problems. First, the decomposition is not unique; for example $\{1\} \oplus \{12, 132\}$ and $\{12\} \oplus \{1, 21\}$ both equal $\{123, 1243\}$, so what should the value of $d_F(\{123, 1243\})$ be?

Secondly and separately, it is not possible to define counting functions that are correct uniformly over all possible inputs $\mathcal{A}$. The examples

$$\{12, 123, 1234\} = \{1, 12\} \oplus \{1, 12\}$$

and

$$\{12, 123, 132, 1243\} = \{1, 12\} \oplus \{1, 21\}$$

demonstrate that there is no counting function that can compute the number of objects in $\mathcal{A}$ of each size from the number of objects in $\mathcal{B}$ and $\mathcal{C}$ of each size, as the first example would require that

$$c_{(F),3}((0, 1, 1), (0, 1, 1)) = 1$$

while the second would require that

$$c_{(F),3}((0, 1, 1), (0, 1, 1)) = 2,$$

i.e., the two examples have different numbers of permutations of length 3 on the left hand side, despite both having the same number of permutations of each length on each component of the right-hand side.

Example 3.4. A two-colored permutation is a permutation in which each of the entries is assigned a color, red or blue. Given two sets of (uncolored) permutations $\mathcal{B}$ and $\mathcal{C}$, we define $\mathcal{B} \circledast \mathcal{C}$ to be the set of two-colored permutations where the red entries form a permutation in $\mathcal{B}$ and the blue entries form a permutation in $\mathcal{C}$. For example, if $132 \in \mathcal{B}$ and $21 \in \mathcal{C}$, then $\overline{1}4\overline{5}\overline{3}2 \in \mathcal{C}$.[6]

We now define a strategy M that we call "colored-merge".

1. The input is a set $\mathcal{A}$ of two-colored permutations. If there exist sets of (ordinary) permutations $\mathcal{B}$ and $\mathcal{C}$ such that $\mathcal{B} \circledast \mathcal{C} = \mathcal{A}$, then the choice of $\mathcal{B}$ and $\mathcal{C}$ is unique and we define $d_M(\mathcal{A}) = (\mathcal{B}, \mathcal{C})$. Otherwise $d_M(\mathcal{A}) = \texttt{DNA}$.

2. In order to form the elements of $\mathcal{A}_n$, it is necessary to know the permutations in $\mathcal{B}$ and $\mathcal{C}$ of sizes $0, \ldots, n$. Therefore, the reliance profile function is $n \mapsto (n, n)$.

3. Each element of $\mathcal{A}_n$ is formed by selecting an element from $\mathcal{B}$ of size j, an element from $\mathcal{C}$ of size $n - j$, and then interleaving the two permutations by selecting which j of the n positions and which j of the n values will be occupied by entries from the red permutation. Thus, the counting functions are

$$c_{M,(n)}((b_0, \ldots, b_n), (c_0, \ldots, c_n)) = \sum_{j=0}^{n} \binom{n}{j}^2 b_j c_{n-j}.$$

Finally, we provide an additional non-example to demonstrate that operations that do not preserve the ability to count are not strategies.

[6]For those who are reading this in black-and-white, the overlined entries are red while the remaining entries are blue.

Non-example 3.5. For a permutation π of length at least 1, let π^- be the permutation that remains after deleting the largest entry of π. Extending this to sets of permutations, define $\mathcal{A}^- = \{\pi^- : \pi \in \mathcal{A}\}$.

Consider the potential decomposition function $d_D(\mathcal{A}) = \mathcal{A}^-$. The equalities

$$d_D(\{123, 132, 312\}) = \{12\} = d_D(\{123\})$$

demonstrate that it is not possible to uniformly compute the counting sequence of $\mathcal{A}$ from the counting sequence of $\mathcal{A}^-$. Therefore we cannot define a strategy with this decomposition function.

When a strategy S is applied to a combinatorial set $\mathcal{A}$ the output of the decomposition function is either $d_S(\mathcal{A}) = \mathtt{DNA}$, in which case no information has been learned, or $d_S(\mathcal{A}) = (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)})$, in which case the strategy has produced what we call a *combinatorial rule*

$$\mathcal{A} \xleftarrow{S} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}),$$

which records the fact that $\mathcal{A}$ can be decomposed into sets $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$ under strategy S. Since S must have a reliance profile function and valid counting functions in order to be considered a strategy, this implies that the counting sequence of $\mathcal{A}$ is a function of the counting sequences of $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$.

3.3 Combinatorial Specifications

In the first chapters of Flajolet and Sedgewick [81], the authors introduce several operations whose input is a tuple of combinatorial sets and whose output is a single combinatorial set. Among these are the disjoint union $\mathcal{A} = \mathcal{B} \sqcup \mathcal{C}$, the Cartesian product $\mathcal{A} = \mathcal{B} \times \mathcal{C}$, the sequence operator $\mathcal{A} = \textsc{Seq}(\mathcal{B})$, and several others. Our decomposition functions are in some sense the "mirror image" of their operators; for them, $\mathcal{B}$ and $\mathcal{C}$ are the inputs to a function that outputs $\mathcal{A} = \mathcal{B} \times \mathcal{C}$, while for us, any set $\mathcal{A}$ may be the input of a decomposition function giving $\mathcal{B}$ and $\mathcal{C}$ as outputs *if such a decomposition is possible*. Although this is not a major deviation, it is this perspective that truly drives the theory of Combinatorial Exploration—combinatorial sets are broken down as much as possible via *decomposition*, rather than built up from atoms, and fingers are crossed that after some number of such decompositions a proof tree can be formed. In the standard viewpoint of symbolic combinatorics, one must figure out from scratch how to express a combinatorial set as the image of some operator; Combinatorial Exploration provides a framework of systematic tools to aid discovering when a combinatorial set can be decomposed into (hopefully) simpler sets.

Flajolet and Sedgwick [81, Definition I.5] additionally define an operator to be *admissible* if the counting sequence of the output set only depends on the counting sequence of the input sets, and a *combinatorial specification* is defined to be a system of operators (more detailed information is given below). They then define and prove the admissibility of several useful operators that they call $+$, $\times$, $\textsc{Seq}$, $\textsc{PSet}$, $\textsc{MSet}$, and $\textsc{Cyc}$, and they call a combinatorial set *constructible* if it is a component of a combinatorial specification made up entirely of these particular operators. By now some readers have surely realized that our notion of a proof tree is similar to the concept of a combinatorial specification, and this section makes that link more precise.

The situation considered by Flajolet and Sedgewick is that you have a given set of objects with known structure, or perhaps defined in terms of a combinatorial specification. Two important issues, whether the specification defines a unique set of objects and whether the resulting system of generating function equations uniquely defines a power series solution, can be confirmed by inspection, and therefore are typically not explicitly addressed. Our approach is to attempt to search for a specification inside a larger set of combinatorial rules, and thus, we need to develop a theory to prove that our approach does not run into these uniqueness issues. Our framework, particularly the consideration of reliance profile functions and counting functions as inherent to a strategy, permits us to study such questions. In Section 4 we define a strategy to be *productive* if it satisfies certain criteria; we then prove that a proof tree composed entirely of rules created by productive strategies is guaranteed to contain sufficient information to uniquely define the counting sequences of the sets involved.

Before describing in detail the correspondence between proof trees and combinatorial specifications, we need to point out a particularly useful kind of strategy. As defined in the previous subsection, a combinatorial rule is a tuple $(\mathcal{A}, (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}), S)$ where $\mathcal{A}$ and $\mathcal{B}^{(i)}$ are combinatorial sets and S is a m-ary strategy. Stylistically, we write

$$\mathcal{A} \xleftarrow{S} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}).$$

Although we did not specifically mention this earlier, it is permissible for a strategy to be 0-ary, or nullary. We call such a strategy a *verification strategy* because it signifies that the counting sequence of the input set is known independently of any structural decomposition. Recall that in the strict framework we have created for strategies, for a given strategy S the counting functions $c_{S,(n)}$ are *independent of the input set* $\mathcal{A}$. As a result, we can define for each combinatorial set $\mathcal{A}$ whose counting sequence is independently known a verification strategy $V_{\mathcal{A}}$ with the following properties:

- ⋄ $d_{V_{\mathcal{A}}}(\mathcal{A}) = ()$ (the 0-tuple) and $d_{V_{\mathcal{A}}}(\mathcal{R}) = \texttt{DNA}$ for all $\mathcal{R} \neq \mathcal{A}$,
- ⋄ $r_{V_{\mathcal{A}}}$ is the function $n \mapsto ()$ (again, the 0-tuple), and
- ⋄ for all n, $c_{V_{\mathcal{A}},(n)}$ is a 0-ary function with output $|\mathcal{A}_n|$.

Of course, one does not want to indiscriminately allow the verification strategies $V_{\mathcal{A}}$ to be applied for all $\mathcal{A}$, or else a trivial proof tree would always be immediately found with no decomposition at all. It is up to the user who is employing Combinatorial Exploration to select which verification strategies to allow—in most cases it makes sense to employ any verification strategy $V_{\mathcal{A}}$ for which the counting sequence for $\mathcal{A}$ is either already known or can be independently calculated. A major benefit of this approach is that once a proof tree for a combinatorial set $\mathcal{R}$ is found, future applications of Combinatorial Exploration can activate the verification strategy $V_{\mathcal{R}}$ to use this knowledge without the extra work of rediscovering the proof tree for $\mathcal{R}$.

In any case, most applications require the use of at least a few verification strategies. In the proof tree for Av(132) in Figure 5, like in nearly all of the proof trees for pattern-avoiding permutation classes, we require the use of $V_{\{\varepsilon\}}$, providing the combinatorial rule

$$\{\varepsilon\} \xleftarrow{V_{\{\varepsilon\}}} (),$$

and in most other proof trees we also require the verification strategy $V_{\{1\}}$ which creates the rule

$$\{1\} \xleftarrow{V_{\{1\}}} ()$$

verifying the set containing the single permutation of length 1.

A *combinatorial specification*, as defined by Flajolet and Sedgewick [81, Definition I.7], is a set of combinatorial rules with the property that every combinatorial set that appears on the right-hand side of a rule appears as the left-hand side of a rule exactly once. The proof tree for $\mathcal{C} = \mathrm{Av}(132)$ in Figure 5, for instance, may be written as the specification

$$\begin{aligned} \mathcal{C} &\xleftarrow{Z} (\{\varepsilon\}, \mathcal{D}) \\ \mathcal{D} &\xleftarrow{F} (\mathcal{C}, \mathcal{C}) \\ \{\varepsilon\} &\xleftarrow{V_{\{\varepsilon\}}} () \end{aligned}$$

Here, Z is the strategy "size-0-or-not" from Example 3.1, while F is the strategy "factor-around-max-entry" from Example 3.2.

Proof trees and combinatorial specifications are two mostly equivalent structures used to represent the same information. Each non-leaf vertex v with children $u_1, \ldots, u_m$ derived from strategy S corresponds to a combinatorial rule

$$D^{(v)} \xleftarrow{S} (D^{(u_1)}, \ldots, D^{(u_m)}),$$

while each leaf w corresponds to either a verification strategy V and rule

$$D^{(w)} \xleftarrow{V} ()$$

or the set $D^{(w)}$ represented by w already appears as a non-leaf vertex in the tree (equivalently, $D^{(w)}$ is already the left-hand side of some rule). One difference between these two structures is that proof trees have a combinatorial set designated as the root, while combinatorial specifications do not (although in practice there is usually one particular set whose enumeration is sought, and this acts as a root). Another difference is that the union of two specifications whose combinatorial sets are disjoint would be considered a specification, while the same is not true for proof trees. Neither of these differences is problematic for the work done here.

The proof tree model gives a more intuitive picture of the structural hierarchy of a combinatorial set as well as better graphical depictions, while the combinatorial specification model is better suited for proving results and describing algorithms. Going forward, we use the two terms largely interchangeably, depending on the context.

As we mentioned in Subsection 2.4 about our successes in applying Combinatorial Exploration to the field of permutation patterns, we employ strategies not discussed in this paper that work with counting functions not just in the size variable n but also in additional variables $k_1, k_2, \ldots$, leading to catalytic variables in the corresponding system of generating function equations. The framework thus requires a multivariate generalization to combinatorial specifications, which we address in future work.

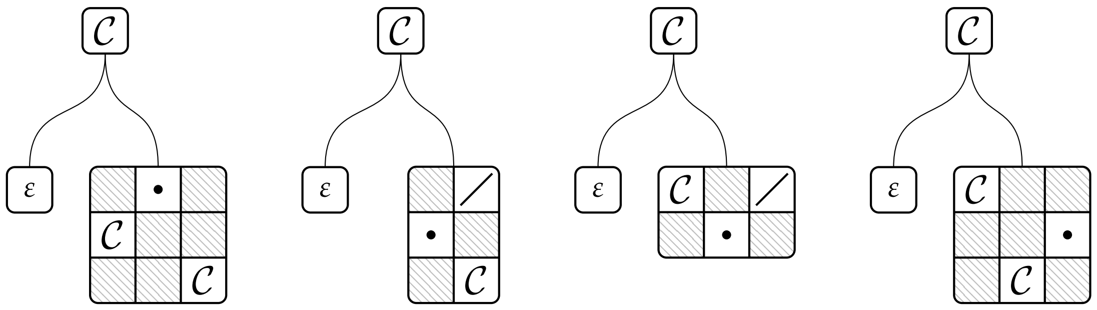


Figure 6: Four different combinatorial rules that result from applying different strategies—in this case, isolating a point in the four different directions—to the same set $\mathcal{C}$.

3.4 Strategic Exploration

In the proof tree for Av(132) shown in Figure 5, the root $\mathcal{C}$ and its two children constitute the first step in a structural description of Av(132): *every permutation in* Av(132) *is either the empty permutation, or contains a maximum entry*. Why did we choose the maximum entry rather than the minimum entry, or the leftmost or rightmost entry? We did so solely because, with the benefit of hindsight, it is that choice that resulted in the nice proof tree we have presented.

In order to discover proof trees like the one for Av(132), the process of Combinatorial Exploration applies to any given combinatorial set many strategies simultaneously.

Figure 6 depicts graphically the four combinatorial rules that result from splitting Av(132) into $\{\varepsilon\}$ and the permutations of length at least 1, and choosing an extreme direction to draw a point.[7] Thus, applying these four strategies (isolating a point in each of the four extreme directions) produces a combinatorial rule. Each rule has the same parent, and each rule has the child $\{\varepsilon\}$, but the second child of each rule is different.

Combinatorial Exploration works by repeatedly applying a collection of strategies, first to the root (the combinatorial set you seek to understand), then to the children of the rules that are produced by these applications, then to the children of those rules, and so on. While it sounds at first like this will suffer from a problem of combinatorial explosion, we shall see that it is surprisingly effective.

We call the ever-growing collection of combinatorial rules that are discovered the *decomposition universe*, and we can depict them pictorially by pushing the graph/tree model just slightly past its useful limit. In a proof tree, the children of a vertex are just the combinatorial sets produced by one single strategy applied to the parent, but in the decomposition universe each parent vertex may have several groups of children, each group coming from the application of a different strategy.

Figure 7 shows an abstract decomposition universe that starts with the root combinatorial set

[7]Note that in the middle two cases, the two non-point cells do not separate into their own rows or columns. These pictures only give a rough idea of the permutations involved, and should not be interpreted too literally. Section 6 introduces a representation called a *tiling* that makes these ideas much more concrete.

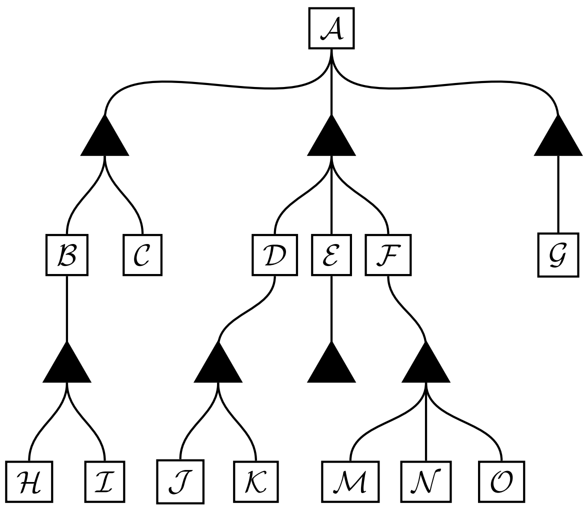


Figure 7: An abstract decomposition universe with seven combinatorial rules.

$\mathcal{A}$. Suppose that some strategy S_1 decomposes $\mathcal{A}$ into $\mathcal{B}$ and $\mathcal{C}$. In the figure, we use a triangular node to group $\mathcal{B}$ and $\mathcal{C}$ together. Suppose then that some other strategy S_2 decomposes $\mathcal{A}$ into $\mathcal{D}$, $\mathcal{E}$, and $\mathcal{F}$, and that some third strategy S_3 decomposes $\mathcal{A}$ into $\mathcal{G}$. There may be other strategies that do not apply to $\mathcal{A}$ at all, and it can be possible that some of the child sets are actually the same set. After attempting to apply all of one's strategies to $\mathcal{A}$, the next step is to try to apply them all to $\mathcal{B}$, perhaps producing some more child groupings as shown in the figure, then apply all of them to $\mathcal{C}$, then $\mathcal{D}$, and so on. It is possible that there are sets to which no strategies apply. Moreover, there may be verification strategies that apply to some of these sets, which would correspond to a grouping of zero children, as can be seen with set $\mathcal{E}$ in the figure.

A simpler and less pictorial but more mathematically manageable way to think of the decomposition universe is simply as a large set of combinatorial rules. The universe shown in Figure 7 corresponds to the set of rules (omitting the strategy names)

$$\begin{array}{llll} \mathcal{A} \leftarrow (\mathcal{B},\mathcal{C}) & \mathcal{A} \leftarrow (\mathcal{D},\mathcal{E},\mathcal{F}) & \mathcal{A} \leftarrow (\mathcal{G}) & \mathcal{B} \leftarrow (\mathcal{H},\mathcal{I}) \\ \mathcal{D} \leftarrow (\mathcal{J},\mathcal{K}) & \mathcal{E} \leftarrow () & \mathcal{F} \leftarrow (\mathcal{M},\mathcal{N},\mathcal{O}). & \end{array}$$

The goal, of course, is to find within the large decomposition universe a proof tree with $\mathcal{A}$ as its root, or equivalently, to find within the large set of combinatorial rules a combinatorial specification (recall, this means a set in which all of the symbols on a right-hand side appear exactly once on a left-hand side) that involves $\mathcal{A}$. It turns out that there is an efficient algorithm to detect the presence of a combinatorial specification among a set of combinatorial rules, as we discuss in the next subsection.

Once a combinatorial specification is detected, Combinatorial Exploration is complete, and that specification can be used to determine whatever enumerative information is desired; Section 4 explores why a specification found by Combinatorial Exploration is guaranteed to contain sufficient information to compute the counting sequences of the sets involved, and Section 5 explains how this is actually done.

Because the framework of Combinatorial Exploration is domain-agnostic, it turns out to be an exceptionally powerful algorithmic approach. Given any domain of combinatorial sets, e.g., permutation classes (Section 6), alternating sign matrices (Section 7), polyominoes (Section 8) and set partitions (Section 9), in order to apply the Combinatorial Exploration framework, one only needs to

- ⋄ decide how to effectively represent the combinatorial objects, and sets of these objects, as a data structure,
- ⋄ implement structural decomposition strategies that discover combinatorial rules,
- ⋄ optionally tune internal parameters, e.g., how often various strategies are applied, in which order, whether depth or breadth is preferred.

Section 10 provides more detailed information about the inner workings of our implementation of Combinatorial Exploration.

3.5 Detecting a Proof Tree In a Decomposition Universe

The process of Combinatorial Exploration produces a large set U of combinatorial rules that we call the decomposition universe and that we may picture as in Figure 7. While every rule in U is a valid application of a strategy, not every rule is useful. Among the rules in U there may or may not be a subset U' that is a combinatorial specification involving the root of the universe (the combinatorial set of interest). If so, and if we can find this subset, then Combinatorial Exploration has successfully done its job, and every combinatorial set that is a part of U' can be enumerated![8]

This raises the question of how a large[9] set U can be efficiently searched for a subset U' that is a combinatorial specification. Viewing the decomposition universe simply as a set of rules, rather than as the graph-like structure in Figure 7, yields a fast and simple algorithm to accomplish this.

The defining property of a combinatorial specification U' is that every set appearing on the right-hand side of a rule also appears on the left-hand side of exactly one rule. It follows that any rule in U that contains a set on the right-hand side that does not appear on any left-hand side cannot be contained in any combinatorial specification $U' \subseteq U$. Algorithm 1 below works by repeatedly removing such rules from U. We prove in Theorem 3.1 that when the algorithm terminates, the remaining set V has the property that every rule in V is in a combinatorial specification within U—that is, V is the union of all combinatorial specifications contained in U. Moreover, if the initial combinatorial set $\mathcal{A}$ that we are trying to enumerate is one of the left-hand sides in V, then there is at least one subset of U' of V that is a combinatorial specification for $\mathcal{A}$. This combinatorial specification U' is the successful output of the Combinatorial Exploration process.

[8]This is true as long as the combinatorial specification has a unique solution, an issue we address in great detail in Section 4.

[9]In our applications, U has occasionally contained over a hundred million rules.

Algorithm 1 Combinatorial Specification Searcher

```
Input: A set of combinatorial rules $U$
Output: The union of all combinatorial specifications contained in $U$

$changed \leftarrow$ True
while $changed$ do
    $changed \leftarrow$ False
    for $\mathcal{A} \xleftarrow{S} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}) \in U$ do
        if any $\mathcal{B}^{(j)}$ is not on the left-hand side of any rule in $U$ then
            $U \leftarrow U \setminus \{\mathcal{A} \xleftarrow{S} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)})\}$
            $changed \leftarrow$ True
        end if
    end for
end while
$V \leftarrow U$
return $V$
```

Theorem 3.1. For any set of combinatorial rules U, the set V returned by Algorithm 1 is equal to the union of all combinatorial specifications that are contained in U.

Proof. Let U be a set of combinatorial rules and let T denote the union of all combinatorial specifications contained in U. Let V be the set output by Algorithm 1 when U is given as input. We will show that $V = T$.

To see that $V \subseteq T$, choose a combinatorial rule $R_1 \in V$. We will show that there is a combinatorial specification $U' \subseteq V$ that contains R_1, thereby ensuring $R_1 \in T$.

Start by defining $U' = \{R_1\}$ and repeat the following steps. Pick a combinatorial set $\mathcal{B}$ that is on the right-hand side of some rule R_2 in U' but is not on any left-hand side. If no such set exists, then U' is a combinatorial specification, and we're finished. Otherwise, we now search for a rule in $V \setminus U'$ that has $\mathcal{B}$ on its left-hand side. There must be some such rule, say R_3, because otherwise Algorithm 1 would have removed R_2 from V. Add R_3 to U', and continue to repeat these steps. Since U is finite, this process is guaranteed to finish in finitely many steps, and as mentioned above, the resulting U' must be a combinatorial specification. Therefore $R_1 \in T$ and the inclusion $V \subseteq T$ is proved.

To see that $T \subseteq V$, let $R \in T$ be a combinatorial rule. Suppose that $R \notin V$. This implies that at some point in the execution of Algorithm 1, R was removed because there was a set on its right-hand side that was not on the left-hand side of any remaining rule. Since $R \in T$, there exists a combinatorial specification $U' \subseteq U$ that contains R, and since R was removed, there must have some rule (possibly R itself) that was the first rule in U' removed by Algorithm 1. This is a contradiction, as every set on the right-hand side of that first-removed rule must still have been on the left-hand side of some rule in U' that had not yet been removed. □

The output of Algorithm 1 has now been proved to be the union of all combinatorial specifications in the universe U of rules. Typically, one wants to obtain just a single combinatorial specification, and Algorithm 2 below describes precisely how to quickly extract from the union of all combinatorial specifications a random specification U' involving the combinatorial set of interest. Algorithm 2 is essentially the procedure used to show that $V \subseteq T$ in the proof of Theorem 3.1. It is also possible to extract, e.g., the smallest specification involving a particular set, but we do not know a fast way to do this.

Algorithm 2 Combinatorial Specification Extractor

```
1: Input: A set of combinatorial rules $V$ output from Algorithm 1 and a combinatorial set $\mathcal{A}$
   on the left-hand side of some rule in $V$
2: Output: A combinatorial specification involving $\mathcal{A}$
3:
4: $seen \leftarrow \varnothing$
5: $spec \leftarrow \varnothing$
6: initialize $queue$
7: push $\mathcal{A}$ to $queue$
8: while $queue$ do
9:     pop $\mathcal{A}'$ from $queue$
10:    choose arbitrarily any one rule of the form $\mathcal{A}' \overset{S}{\leftarrow} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}) \in V$
11:    $spec \leftarrow spec \cup \{\mathcal{A}' \overset{S}{\leftarrow} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)})\}$
12:    $seen \leftarrow seen \cup \{\mathcal{A}'\}$
13:    for $\mathcal{B}$ in $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$ do
14:        if $\mathcal{B}$ not in $seen$ then
15:            push $\mathcal{B}$ to $queue$
16:        end if
17:    end for
18: end while
19: return $spec$
```

Our software to perform Combinatorial Exploration is available on GitHub. The main repository can be found at `https://github.com/PermutaTriangle/comb_spec_searcher` [22] and a second repository that implements Combinatorial Exploration for the domain of permutations (see Section 6) can be found at `https://github.com/PermutaTriangle/Tilings` [21]. Despite the simplicity behind the idea of Combinatorial Exploration, we have had to develop a number of novel algorithms to produce an efficient implementation. Section 10 briefly describes just a few of the computational challenges that needed to be addressed.

4. Productive Proof Trees and Combinatorial Specifications

Although combinatorial specifications have appeared often in the literature, there does not seem to be any uniform treatment of the inconvenient fact that some combinatorial specifications convey no enumerative information—they cannot be used to count the number of objects of size n in each combinatorial set. We call such a combinatorial specification *trivial*.

Consider, for example, the combinatorial specification

$$\begin{aligned}\mathcal{A} &\xleftarrow{S_1} (\mathcal{B},\mathcal{C})\\ \mathcal{B} &\xleftarrow{S_2} (\{\varepsilon\},\mathcal{C})\\ \mathcal{C} &\xleftarrow{S_3} (\{\},\mathcal{B})\end{aligned}$$

where ε is a combinatorial object of size 0, S_1 and S_3 are any strategies identifying that $|\mathcal{A}_n| = |\mathcal{B}_n| + |\mathcal{C}_n|$ and $|\mathcal{C}_n| = |\{\}_n| + |\mathcal{B}_n|$, and S_2 is a strategy identifying that

$$|\mathcal{B}_n| = \sum_{j=0}^{n} |\{\varepsilon\}_j||\mathcal{C}_{n-j}|.$$

Upon simplification, this system of recurrences becomes

$$\begin{aligned}|\mathcal{A}_n| &= |\mathcal{B}_n| + |\mathcal{C}_n|\\ |\mathcal{B}_n| &= |\mathcal{C}_n|\\ |\mathcal{C}_n| &= |\mathcal{B}_n|,\end{aligned}$$

which of course cannot be used to calculate any terms in the counting sequences of the combinatorial sets.

Although triviality is obvious in this simple example, avoiding triviality becomes more delicate as strategies become more intricate.

One might worry that in order to prove that a combinatorial specification is *productive* (that is, non-trivial), it could be necessary to examine the global structure of its corresponding proof tree. Surprisingly, this is not the case. In this section, we present a set of local conditions that can be placed on strategies, and we call a strategy satisfying these a *productive strategy*. We then prove that when each combinatorial rule in a specification is generated by a productive strategy, the result is a productive combinatorial specification. We believe that this general result is, on its own, a significant new contribution to the literature on combinatorial specifications.

4.1 Reliance Graphs

Suppose P is a proof tree whose root is the combinatorial set $\mathcal{C}$. In order to compute the size of $\mathcal{C}_{10}$, we can imagine "asking" the root: "How many elements do you contain of size 10?" Suppose that the children of $\mathcal{C}$ are $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$ derived by the strategy S, i.e., that there is a combinatorial rule

$$\mathcal{C} \xleftarrow{S} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}).$$

In order for the root $\mathcal{C}$ to "answer" our question about the number of elements of size 10, the information contained in the strategy S tells it to ask its children $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$ for the number of elements they each contain of particular sizes (according to the reliance profile function), and how to combine those numbers to get the answer (according to the counting functions). To be more explicit, suppose $r_S(10) = (r_S^{(1)}(10), \ldots, r_S^{(m)}(10))$. Then, our request for $|\mathcal{C}_{10}|$ leads to requests for $|\mathcal{B}_j^{(1)}|$ for $j \in \{0, \ldots, r_S^{(1)}(10)\}$, $|\mathcal{B}_j^{(2)}|$ for $j \in \{0, \ldots, r_S^{(2)}(10)\}$, etc., and those quantities are passed as input into $c_{S,(10)}$, whose output is then $|\mathcal{C}_{10}|$ as desired. Each request for $|\mathcal{B}_j^{(i)}|$ is

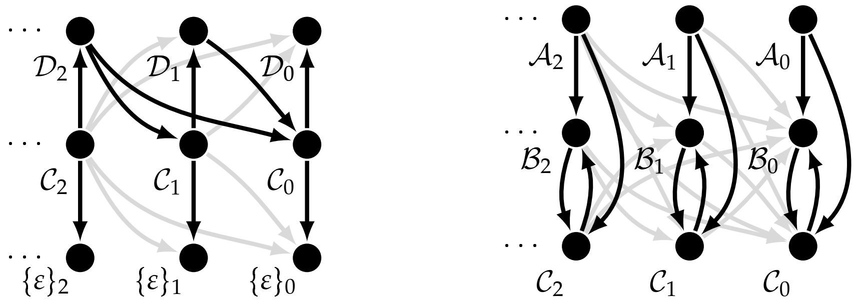


Figure 8: Partial depictions of two reliance graphs.

handled recursively in the same manner—if $|\mathcal{B}_j^{(i)}|$ has already been calculated then the answer is known; if there is a verification strategy $V_{\mathcal{B}^{(i)}}$ and the corresponding rule $\mathcal{B}^{(i)} \xleftarrow{V_{\mathcal{B}^{(i)}}} ()$, then $|\mathcal{B}_j^{(i)}|$ is already known; otherwise, $\mathcal{B}^{(i)}$ is decomposed via a strategy into its children, and the recursion continues.

If this process terminates, then the root $\mathcal{C}$ is able to answer our question, and we are told the number of elements in $\mathcal{C}$ of size 10. We aim to characterize when this process terminates, and when it does not. In order to do so, we extend the notion of reliance profile functions from strategies to specifications (and, equivalently, proof trees). The *reliance graph* R of a combinatorial specification C is an infinite directed graph defined as follows. For each combinatorial set $\mathcal{B}^{(i)}$, there is an infinite family of vertices $\{\mathcal{B}_n^{(i)} : n \in \mathbb{N}\}$. There is a directed edge from $\mathcal{B}_{j_1}^{(i_1)}$ to $\mathcal{B}_{j_2}^{(i_2)}$ if $\mathcal{B}^{(i_2)}$ is on the right-hand side of the rule whose left-hand side is $\mathcal{B}^{(i_1)}$, and the reliance profile function r_S dictates that $\mathcal{B}_{j_1}^{(i_1)}$ relies on $\mathcal{B}_{j_2}^{(i_2)}$, i.e., if $\mathcal{B}^{(i_2)}$ is the ℓth element of $d_S(\mathcal{B}^{(i_1)})$, then $j_2 \leqslant r_S^{(\ell)}(j_1)$. Informally a directed edge from $\mathcal{B}_{j_1}^{(i_1)}$ to $\mathcal{B}_{j_2}^{(i_2)}$ conveys that in order to find the count $|\mathcal{B}_{j_1}^{(i)}|$ it is required to first calculate $|\mathcal{B}_{j_2}^{(i_2)}|$.

We take this opportunity to examine a couple of examples, first looking once again at the specification for Av(132) on page 21 using strategies Z, F, $V_{\{\varepsilon\}}$, and $V_{\{1\}}$ described in Subsection 3.3.

The strategy Z applied to the root carries the reliance profile function $n \mapsto (n, n)$, the strategy F carries the reliance profile function $n \mapsto (n-1, n-1)$, while the other two strategies are verification strategies and carry the reliance profile function $n \mapsto ()$. An abbreviated portion of the reliance graph for this specification is shown on the left in Figure 8. In this graph, each vertex $\mathcal{C}_n$ is the source of a directed edge to each of $\{\varepsilon\}_i$ and $\mathcal{D}_i$ for $i \leqslant n$, each vertex $\mathcal{D}_n$ is the source of a directed edge to each of $\mathcal{C}_i$ for $i \leqslant n-1$, and each vertex $\{\varepsilon\}_n$ is the source of no directed edges (being the parent of a verification strategy). In an attempt to make the graph more readable, we have drawn edges in gray whose reliance is not actually used in the counting formulas, e.g., although $|\mathcal{D}_1|$ is technically an input into the counting formula for $|\mathcal{C}_2|$, it is not used as part of the computation in that counting formula.

On the other hand, the trivial combinatorial specification at the beginning of this section has the reliance graph that is partially shown on the right in Figure 8. It is clear from this reliance graph that the recursive enumeration procedure that we have described fails. When the vertex $\mathcal{B}$ is asked for the number of its elements of size 2, it poses the same question to $\mathcal{C}$, which in turns poses the same question to $\mathcal{B}$, *ad infinitum*.

The characteristic of a reliance graph that determines whether our recursive enumeration procedure is effective is the absence or presence of infinite directed walks; the reliance graph on the right in Figure 8 possesses infinite walks (e.g., $\mathcal{A}_2 \to \mathcal{B}_2 \to \mathcal{C}_2 \to \mathcal{B}_2 \to \mathcal{C}_2 \to \cdots$), while the reliance graph on the left does not. In this subsection, we prove that if the reliance graph of a combinatorial specification has no infinite walks, then the combinatorial specification uniquely determines the counting sequences of all sets involved.

We start by discussing what it means for a combinatorial specification to uniquely determine its counting sequences. For each combinatorial rule $\mathcal{A} \xleftarrow{S} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)})$ and each $n \in \mathbb{N}$ we can obtain a symbolic equation relating $|\mathcal{A}_n|$ with the various quantities $|\mathcal{B}_j^{(i)}|$ dictated by the reliance profile function of S. It is convenient here to take the "combinatorial specification" perspective instead of the "proof tree" perspective. Let P be a combinatorial specification involving N combinatorial sets $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(N)}$. This implies that there are N combinatorial rules:

$$\begin{aligned}
\mathcal{B}^{(1)} &\xleftarrow{S_1} (\mathcal{B}^{(i_{1,1})}, \mathcal{B}^{(i_{1,2})}, \ldots, \mathcal{B}^{(i_{1,m_1})}) \\
\mathcal{B}^{(2)} &\xleftarrow{S_2} (\mathcal{B}^{(i_{2,1})}, \mathcal{B}^{(i_{2,2})}, \ldots, \mathcal{B}^{(i_{2,m_2})}) \\
&\vdots \\
\mathcal{B}^{(N)} &\xleftarrow{S_N} (\mathcal{B}^{(i_{N,1})}, \mathcal{B}^{(i_{N,2})}, \ldots, \mathcal{B}^{(i_{N,m_N})}).
\end{aligned}$$

To each of these rules we associate an infinite sequence of equations, one for each $n \in \mathbb{N}$, derived from the corresponding counting function:

$$b_n^{(j)} = c_{S_j,(n)}\left(\left(b_0^{(i_{j,1})}, b_1^{(i_{j,1})}, \ldots, b_{r_{S_j}^{(1)}(n)}^{(i_{j,1})}\right), \ldots, \left(b_0^{(i_{j,m_j})}, b_1^{(i_{j,m_j})}, \ldots, b_{r_{S_j}^{(m_j)}(n)}^{(i_{j,m_j})}\right)\right).$$

For example, the strategy Z from Example 3.1, when producing a combinatorial rule $\mathcal{B}^{(1)} \xleftarrow{Z} (\mathcal{B}^{(2)}, \mathcal{B}^{(3)})$ produces symbolic equations

$$b_0^{(1)} = b_0^{(2)} + b_0^{(3)}, \qquad b_1^{(1)} = b_1^{(2)} + b_1^{(3)}, \qquad b_2^{(1)} = b_2^{(2)} + b_2^{(3)}, \qquad \ldots.$$

The strategy F from Example 3.2, when producing a combinatorial rule $\mathcal{B}^{(1)} \xleftarrow{F} (\mathcal{B}^{(2)}, \mathcal{B}^{(3)})$ produces symbolic equations

$$b_0^{(1)} = 0, \qquad b_1^{(1)} = b_0^{(2)} b_0^{(3)}, \qquad b_2^{(1)} = b_0^{(2)} b_1^{(3)} + b_1^{(2)} b_0^{(3)}, \qquad \ldots.$$

For a verification strategy, the right-hand sides of the symbolic equations are just constants. For instance, the verification strategy V_ε used in the combinatorial specification for $\mathrm{Av}(132)$ produces symbolic equations

$$|\{\varepsilon\}_0| = 1, \qquad |\{\varepsilon\}_1| = 0, \qquad |\{\varepsilon\}_2| = 0, \qquad \ldots.$$

We are now ready to state an important definition that will be the focus of the remainder of this section.

Definition 4.1. A proof tree involving N combinatorial sets and its corresponding combinatorial specification are called *productive* if the infinite system of equations produced by the counting functions has a unique solution in $\left(\mathbb{C}^{\mathbb{N}}\right)^N$.

Productivity is clearly the property that one wants a combinatorial specification to possess, otherwise the specification may not be useful. The following theorem shows that the productivity of a specification can be checked by examining its reliance graph.

Theorem 4.1. Let P be a proof tree (or the corresponding specification) involving combinatorial sets $\mathcal{B}^{(1)}, \mathcal{B}^{(2)}, \ldots, \mathcal{B}^{(N)}$ and whose reliance graph contains no infinite directed walks. Let $\mathcal{S}(P)$ be the system of equations in the indeterminates $\{b_j^{(i)} \;:\; j \in \mathbb{N},\; 1 \leqslant i \leqslant N\}$. There exists a unique solution to the system

$$\left(\left(\widetilde{b}_0^{(1)}, \widetilde{b}_1^{(1)}, \ldots\right), \left(\widetilde{b}_0^{(2)}, \widetilde{b}_1^{(2)}, \ldots\right), \ldots, \left(\widetilde{b}_0^{(N)}, \widetilde{b}_1^{(N)}, \ldots\right)\right) \in \left(\mathbb{C}^{\mathbb{N}}\right)^N.$$

In other words, P is a productive proof tree.

Proof. By construction, one solution to the system $\mathcal{S}(P)$ is the one which enumerates the combinatorial sets in the proof tree. We call this solution

$$\widetilde{b} = \left(\left(\widetilde{b}_0^{(1)}, \widetilde{b}_1^{(1)}, \ldots\right), \left(\widetilde{b}_0^{(2)}, \widetilde{b}_1^{(2)}, \ldots\right), \ldots, \left(\widetilde{b}_0^{(N)}, \widetilde{b}_1^{(N)}, \ldots\right)\right).$$

Suppose there were a different solution

$$\widetilde{d} = \left(\left(\widetilde{d}_0^{(1)}, \widetilde{d}_1^{(1)}, \ldots\right), \left(\widetilde{d}_0^{(2)}, \widetilde{d}_1^{(2)}, \ldots\right), \ldots, \left(\widetilde{d}_0^{(N)}, \widetilde{d}_1^{(N)}, \ldots\right)\right).$$

Let R be the reliance graph of P. For any vertex v in R, define $\operatorname{rank}(v)$ to be the length of the longest walk in R that starts at v.

We claim that $\operatorname{rank}(v)$ exists and is finite. To justify this, we must rule out the possibility that there are infinitely many walks that start at v with lengths that are arbitrarily large, but finite. The outdegree of any vertex in a reliance graph is finite, and so if there were infinitely many walks that started at v_1, then there would be at least one child of v_1, say v_2 from which infinitely many walks started. Carrying on in this way, we can form a walk $v_1, v_2, v_3, \ldots$, which has infinite length, contradicting our assumption that R has no infinite walks. Therefore, finitely many walks start from v, and thus $\operatorname{rank}(v)$ is finite.

Note that each indeterminate $b_j^{(i)}$ corresponds to one vertex in R, namely the one labeled $\mathcal{B}_j^{(i)}$. The vertices that are leaves in the reliance graph are precisely those for which $\mathcal{B}^{(i)}$ is on the left-hand side of a verification rule, and the corresponding equation in $\mathcal{S}(P)$ with $b_j^{(i)}$ on the left-hand side has as its right-hand side an explicit natural number.

Now we consider the two distinct solutions $\widetilde{b}$ and $\widetilde{d}$. Among all pairs $\widetilde{b}_j^{(i)}, \widetilde{d}_j^{(i)}$ where the solutions differ (that is, $\widetilde{b}_j^{(i)} \neq \widetilde{d}_j^{(i)}$), choose one pair $\widetilde{b}_J^{(I)}, \widetilde{d}_J^{(I)}$ that is minimal with respect to the rank of the corresponding vertex in R (the one labeled $\mathcal{B}_J^{(I)}$). That minimal rank is guaranteed to be at least one, because the rank zero vertices are leaves, and by the discussion above it is clear that the solutions $\widetilde{b}$ and $\widetilde{d}$ must agree at all indeterminates corresponding to leaves.

There is a single equation in $\mathcal{S}(P)$ with $b_J^{(I)}$ on the left-hand side. The right-hand side involves some nonempty set of variables of the form $b_i^{(j)}$ (we will call these variables the *children* of $b_I^{(J)}$ for the moment). The vertices in R that are labeled by these children (the *child vertices*) are precisely the ones at the other end of all outgoing edges from the vertex labeled $\mathcal{B}_J^{(I)}$, and therefore the rank of each child vertex must be strictly less than the rank of their parent vertex $\mathcal{B}_J^{(I)}$.

By our minimality assumption, it follows that the solutions $\widetilde{b}$ and $\widetilde{d}$ agree for each of the child variables. But this cannot be, since $\widetilde{b}_J^{(I)}$ and $\widetilde{d}_J^{(I)}$ are each computed from the values of these child indeterminates in the same manner, and thus this would imply $\widetilde{b}_J^{(I)} = \widetilde{d}_J^{(I)}$. □

4.2 Productive Strategies

We have just proved that any specification whose reliance graph contains no infinite directed walks is productive; that is, it contains sufficient information to uniquely determine the counting sequences of all of its combinatorial sets. It is natural to now ask how one can determine when the reliance graph of a specification contains no infinite directed walks.

Given any particular concrete combinatorial specification, it can surely be determined by hand (though perhaps with great effort for large specifications) whether the reliance graph contains infinite directed walks. However, in order for Combinatorial Exploration to be effective, we want to be certain ahead of time that any specification produced will be productive. This is not just to avoid the need to check an output specification, but also because the efficient method we described in Section 3.5 is not capable of simply "ignoring" an unproductive specification and continuing to search for a productive one.

It is perhaps surprising that it is possible to place local restrictions on individual strategies that then guarantee that any specification that employs only those strategies is itself productive. To accomplish this, we introduce the concept of a *productive strategy*.

Definition 4.2. We call an m-ary strategy S a *productive strategy* if the following two conditions hold for all combinatorial sets $\mathcal{A}$ with corresponding decomposition $d_S(\mathcal{A}) = (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)})$, and for all $i \in \{1, \ldots, m\}$.

1. For all $N \in \mathbb{N}$, if $\mathcal{A}_N$ relies on $\mathcal{B}_j^{(i)}$, then $j \leqslant N$.
2. If $\mathcal{A}_N$ relies on $\mathcal{B}_N^{(i)}$ for some $N \in \mathbb{N}$, then
   (a) $|\mathcal{A}_n| \geqslant |\mathcal{B}_n^{(i)}|$ for all $n \in \mathbb{N}$, and
   (b) $|\mathcal{A}_\ell| > |\mathcal{B}_\ell^{(i)}|$ for some $\ell \in \mathbb{N}$.

As before, we use the phrase "$\mathcal{A}_n$ relies on $\mathcal{B}_j^{(i)}$" or the diagram $\mathcal{A}_n \to \mathcal{B}_j^{(i)}$ as a simplified way of stating the formal information contained in the reliance profile function of S, namely that $j \leqslant r_S^{(i)}(n)$.

In other words, each reliance either moves from a larger object size to a smaller object size ($N > j$), in which case $|\mathcal{B}_j^{(i)}|$ can be smaller, larger, or the same size as $|\mathcal{A}_N|$, or stays at the same size of object ($N = j$), in which case the child set cannot be larger than the parent at *any* size, and must actually be properly smaller for at least one size.

Although these conditions may at first seem restrictive, it turns out that they are satisfied by many natural strategies. We now prove that the reliance graph of any proof tree that is composed entirely of productive strategies has no infinite directed walks.

Theorem 4.2. Let P be a proof tree, or the equivalent combinatorial specification, composed entirely of rules derived from productive strategies. Then the reliance graph of P has no infinite directed walks.

Proof. Let R be the reliance graph of P. Suppose toward a contradiction that there is an infinite walk W. The set of vertices involved in W is finite—indeed, the only vertices that can be involved in a walk W that starts at vertex $\mathcal{A}_\ell$ are those of the form $\mathcal{D}_j$ where $\mathcal{D}$ is any combinatorial set involved in P and $j \leqslant \ell$. As W is an infinite walk but involves only a finite set of vertices, W must contain a cycle

$$\mathcal{A}_N \to \mathcal{E}^{(1)}_{N_1} \to \mathcal{E}^{(2)}_{N_2} \to \cdots \to \mathcal{E}^{(m)}_{N_m} \to \mathcal{A}_N.$$

Condition 1 further implies that the sizes of objects involved in this cycle are all the same, i.e., $N = N_1 = N_2 = \cdots = N_m$. Hence,

$$\mathcal{A}_N \to \mathcal{E}^{(1)}_{N} \to \mathcal{E}^{(2)}_{N} \to \cdots \to \mathcal{E}^{(m)}_{N} \to \mathcal{A}_N.$$

Since we cannot have $\mathcal{A}_N \to \mathcal{A}_N$ (this would fail Condition 2(b)), we must have $m \geqslant 1$.

It now follows from Condition 2(a) that

$$|\mathcal{A}_n| \geqslant |\mathcal{E}^{(1)}_n| \geqslant |\mathcal{E}^{(2)}_n| \geqslant \cdots \geqslant |\mathcal{E}^{(m)}_n| \geqslant |\mathcal{A}_n|$$

and thus

$$|\mathcal{A}_n| = |\mathcal{E}^{(1)}_n| = |\mathcal{E}^{(2)}_n| = \cdots = |\mathcal{E}^{(m)}_n| = |\mathcal{A}_n|$$

for all $n \in \mathbb{N}$. As a result $\mathcal{A}$ is equinumerous to $\mathcal{E}^{(1)}$, contradicting Condition 2(b). □

Combining Theorems 4.1 and 4.2 gives the following key result.

Theorem 4.3. Let P be a proof tree, or the equivalent combinatorial specification, composed entirely of rules derived from productive strategies. Then P is productive, i.e., the infinite system of equations derived from its counting functions has a unique solution.

Furthermore, the discussion at the beginning of Section 4.1 demonstrates how the unique counting sequences for a productive specification can actually be computed.

4.3 Equivalence Strategies

The formalization of productive strategies and proof trees has one significant deficiency. An *equivalence strategy* is any 1-ary strategy that identifies that two combinatorial sets are equinumerous. More precisely, S is an equivalence strategy if whenever $d_S(\mathcal{A}) = \mathcal{B}$, we have $|\mathcal{A}_n| = |\mathcal{B}_n|$ for all n. The reliance profile function for any equivalence strategy can be assumed to be $r_S(n) = (n)$.

The following sections demonstrate that equivalence strategies are powerful tools for identifying structural decompositions of combinatorial sets, yet according to our definition they are clearly not productive strategies—they always fail Condition 2 of 4.2.

It is easy to see why equivalence strategies lead to non-productive proof trees. Suppose some equivalence strategy S produces the rule $\mathcal{A} \overset{S}{\leftarrow} \mathcal{B}$ and some other equivalence strategy T produces the rule $\mathcal{B} \overset{T}{\leftarrow} \mathcal{A}$. These two rules together form a combinatorial specification (because every set on a right-hand side appears exactly once on a left-hand side), although all we can conclude is that $|\mathcal{A}_n| = |\mathcal{B}_n|$. There is not sufficient information to compute the counting sequence for either $\mathcal{A}$ or $\mathcal{B}$.

One possible solution to this issue is to ensure that whenever a rule of the form $\mathcal{A} \overset{S}{\leftarrow} \mathcal{B}$ for an equivalence strategy S exists in a proof tree, no rule $\mathcal{B} \leftarrow \mathcal{A}$ is permitted. This is impractical for several reasons. First, while Combinatorial Exploration is expanding the universe of combinatorial rules, it is not clear which direction of a rule will prove to be more useful when searching for a proof tree. More importantly, significant power is lost by ignoring that the sets are equinumerous and thus that their counting sequences can be substituted for each other as needed.

There is another solution that is simpler and yet preserves all information provided by the equivalence strategies. Up to this point, we have defined a combinatorial rule in a specification (equivalently, a parent-children relationship in a proof tree) to be of the form

$$\mathcal{A} \overset{S}{\leftarrow} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}),$$

where $\mathcal{A}$ and the $\mathcal{B}^{(i)}$ are combinatorial sets. To generalize this, we sort all combinatorial sets involved into equivalence classes in the following way. First, we say that $\mathcal{A}$ is related to $\mathcal{B}$ if $\mathcal{A} = \mathcal{B}$ or if there is an equivalence strategy S that either produces the rule $\mathcal{A} \overset{S}{\leftarrow} \mathcal{B}$ or the rule $\mathcal{B} \overset{S}{\leftarrow} \mathcal{A}$. Then we take the transitive closure of this relation to form the equivalence relation $\mathcal{R}$. Two combinatorial sets in the same equivalence class of $\mathcal{R}$ are guaranteed to be equinumerous, but it is not necessarily true that any two equinumerous combinatorial sets are in the same equivalence class.

To mitigate the productivity issue caused by equivalence strategies, we henceforth consider the vertices of a proof tree to represent not combinatorial sets, but equivalence classes of combinatorial sets from $\mathcal{R}$.[10] Equivalently, the symbols on the left- and right-hand sides of combinatorial rules do not represent combinatorial sets, but equivalence classes of combinatorial sets. The combinatorial rules produced by equivalence strategies themselves are not added to the ever-growing set of combinatorial rules, since they are not productive.

Often the identification of two sets as equinumerous can lead to the discovery of proof trees much smaller than might otherwise be found. Many specific examples are seen in later sections, but we give a generic example here. Suppose that while in the process of Combinatorial Exploration we have produced the following rules (in which the symbols on each side are still

[10]An important feature of the Combinatorial Exploration framework is that as strategies are repeatedly applied, we may continue to add new sets to equivalence classes or merge two existing equivalence classes together.

single combinatorial sets):

$$\begin{aligned}\mathcal{A} &\xleftarrow{S} (\mathcal{B},\mathcal{C})\\ \mathcal{B} &\xleftarrow{V_{\mathcal{B}}} ()\\ \mathcal{D} &\xleftarrow{E} (\mathcal{C})\\ \mathcal{D} &\xleftarrow{T} (\mathcal{A},\mathcal{B})\end{aligned}$$

where S and T are productive strategies, $V_{\mathcal{B}}$ is a verification strategy, and E is an equivalence strategy. This set of rules does not contain a combinatorial specification, let alone a productive one.

However, when grouping the combinatorial sets into equivalence classes as described above, we obtain the productive combinatorial specification

$$\begin{aligned}\{\mathcal{A}\} &\xleftarrow{S} (\{\mathcal{B}\},\{\mathcal{C},\mathcal{D}\})\\ \{\mathcal{B}\} &\xleftarrow{V_{\mathcal{B}}} ()\\ \{\mathcal{C},\mathcal{D}\} &\xleftarrow{T} (\{\mathcal{A}\},\{\mathcal{B}\}).\end{aligned}$$

5. Transfer Tools

Finding a combinatorial specification for a set is often an intermediate step in the quest to fully understand the set. This brief section outlines several ways in which a combinatorial specification can be transferred into other products including polynomial-time counting algorithms, systems of equations for the generating function, and uniform random sampling routines.

5.1 Polynomial-Time Counting Algorithms

A polynomial-time counting algorithm for a combinatorial set $\mathcal{A}$ is a routine that can calculate $|\mathcal{A}_n|$ in polynomially-many (in n) arithmetic operations. Which operations are included in this definition can vary according to one's preferences or needs; here we choose addition, subtraction, multiplication, and division. Brute force computation of $|\mathcal{A}_n|$ takes exponential time or more in many applications, but often possession of a combinatorial specification enables fast computation of many terms in the counting sequence.

A combinatorial strategy S is called a *polynomial-time strategy* if, for all $n \in \mathbb{N}$, computing the output of the counting function $c_{S,(n)}$ requires only polynomially-many (in n) arithmetic operations. All strategies discussed in this article are polynomial-time strategies.

If C is a combinatorial specification made up entirely of productive polynomial-time strategies, then the routine discussed in Section 4.1 for computing terms in the counting sequence of any set involved in C takes only polynomially-many arithmetic operations.

5.2 Generating Functions

Possession of a combinatorial specification also often allows one to write down a system of equations satisfied by the generating functions of each of the combinatorial sets involved.

These systems can then sometimes be solved to yield the generating function for the specific set of interest.

To produce such a system of equations, one must associate to each strategy S used a function g_S that describes how the generating functions of each input set can be combined to produce the generating function of the output set. For example, suppose S is a strategy with decomposition function $d_S(\mathcal{A}) = (\mathcal{B}^{(1)}, \mathcal{B}^{(2)})$ and counting functions $a_n = b_n^{(1)} + b_n^{(2)}$, then we can assign $g_S(B_1, B_2) = B_1 + B_2$. Now, if $A(x)$, $B^{(1)}(x)$, and $B^{(2)}(x)$ are the generating functions of $\mathcal{A}$, $\mathcal{B}^{(1)}$, and $\mathcal{B}^{(2)}$, then we have the equality

$$A(x) = g_S(B^{(1)}(x), B^{(2)}(x)).$$

Another common example is a binary strategy S with counting function $a_n = \sum_{k=0}^{n} b_k^{(1)} b_{n-k}^{(2)}$, which corresponds to the equation $g_S(B_1, B_2) = B_1 B_2$.

If V is a verification strategy, then we consider g_V to simply have no inputs, while the output is the generating function for any combinatorial set that V verifies (recall that for a fixed verification strategy V, all sets to which it applies must, by definition, have the same counting sequence).

In this introductory work, many of the strategies we describe lead to generating function equations that are either sums, products, or explicit generating functions. The reader should not get the impression that this is a limitation of the Combinatorial Exploration framework. It is simply a result of the fact that we have chosen to defer the introduction of more complicated strategies, and a multivariate generalization of the entire framework, to future work.

Still, we cannot resist showing a morsel here. It is often desirable to consider combinatorial sets indexed not just by size n, but by some additional statistic k. For example, $\mathcal{A}_{n,k}$ could be the set of binary words of length n that contain exactly k occurrences of the symbol 1. In many cases, these additional statistics are actually required in order to find a specification at all. Accordingly, the whole theory of strategies (particularly, their reliance profile functions and counting functions), the concepts of reliance graphs and productivity, and the transfer tools described in this section all require suitable generalization. The result is that systems of equations now involve multiple variables in complicated ways, with equations such as

$$A(x, y) = \frac{yB(x, y) - B(x, 1)}{y - 1}$$

and

$$C(x, w, y, z) = D\left(x, \frac{wy}{z}, w\right),$$

leading to systems that either require advanced methods to solve (e.g., extensions of the ideas of Bousquet-Mélou and Jehanne [44]) or that we do not know how to solve at all!

5.3 Random Sampling

It is computationally difficult in general to sample an object of size n from a combinatorial set $\mathcal{A}$ uniformly at random because the brute force method of doing so requires computing

$\mathcal{A}_n$ explicitly. However, the reader will not be surprised that having a combinatorial specification involving the set $\mathcal{A}$ often leads to random sampling routines that can be performed in polynomial time. Although our implementation of Combinatorial Exploration [22] is already capable of performing random sampling, we defer the details, including which properties that strategies in a specification must have for this to be possible, to a later article.

Figure 9 demonstrates this ability by showing heatmaps for all classes up to symmetry defined by avoiding two patterns of length 4 (except for $\text{Av}(1234, 4321)$, which contains a finite number of permutations), and three of the seven symmetry classes defined by avoiding one pattern of length 4. Each heatmap is formed by sampling one million permutations of length 300 uniformly at random from the class and forming a picture 300 pixels wide and tall such that the darkness of the pixel at location (i, j) corresponds to how many of the one million sampled permutations have an entry at index i and value j (i.e., $\pi(i) = j$). Lighter pixels indicated fewer such entries, and darker, more. Higher resolution pictures can be viewed on the pages for each corresponding class on the PermPAL website [6].

Less formally, these pictures can be thought of as taking the one million sampled permutations, forming their plots as described in Section 2, making them mostly transparent, and stacking them on top of each other. In this way, a heatmap conveys information about what permutations in a particular class tend to "look like". This concept has been studied much more formally by Miner and Pak [119], Hoffman, Rizzolo, and Slivken [89], and many others.

6. Applying Combinatorial Exploration to Permutation Patterns

This and the several following sections demonstrate the potency of Combinatorial Exploration by applying it to a number of popular combinatorial objects. For each new domain to which Combinatorial Exploration is to be applied we must do the following.

1. Provide a representation of combinatorial sets in this domain.
2. Invent strategies that decompose (some of the) combinatorial sets into other combinatorial sets.
3. Prove the productivity of any non-equivalence strategy.

As will become clear over the next few sections, there are choices to be made in the first two steps that may determine the efficacy of Combinatorial Exploration. Consider for example the combinatorial domain explored in this section, pattern-avoiding permutation classes (see Section 2 for the relevant definitions). Before describing strategies that decompose combinatorial sets, we must decide what the objects in the combinatorial sets actually are. The most obvious choice is that the objects should be precisely the things we are counting: permutations. It turns out that such a simple structure does not lend itself to discovering or describing decomposition strategies.

Taking some inspiration from the literature, we will define *gridded permutations* below as a generalization of permutations with a geometric flavor. Our representations of the combinatorial domains in later sections also have a geometric bent. This is certainly not a requirement to employ Combinatorial Exploration, but it is our hope that these sections convey how a geometric structure often leads to effective decomposition strategies that are easier to describe. The task of making and implementing design decisions such as choosing representations for objects and

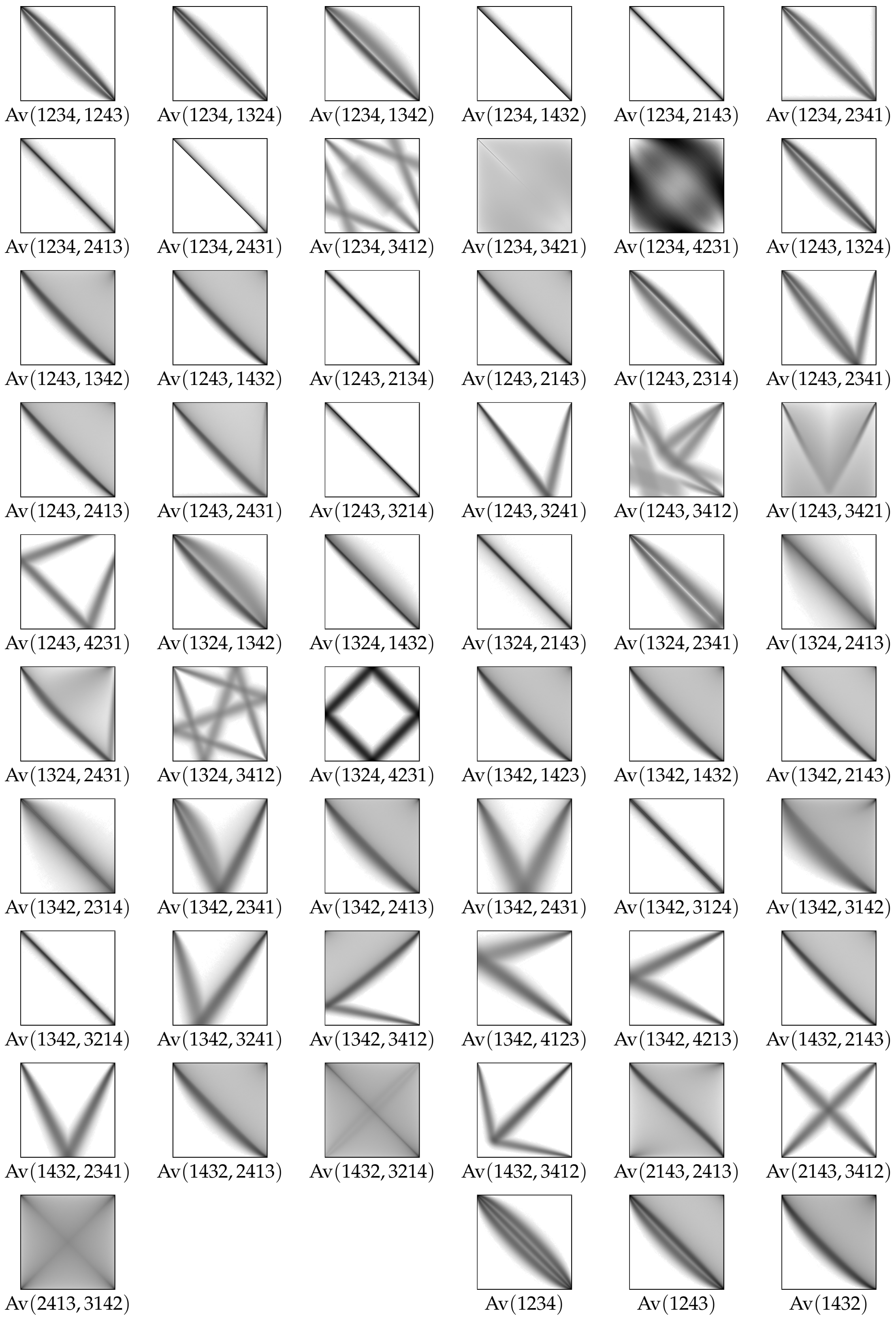


Figure 9: Heatmaps of 58 permutation classes.

devising structural strategies is the portion of using Combinatorial Exploration where knowledge and experience in the given domain is critical. Once this task is done, one can sit back and allow Combinatorial Exploration to enumerate combinatorial sets fully rigorously and automatically.

Of the combinatorial domains explored in this article, we have spent by far the most time with permutation patterns. While this section demonstrates the depth of success that Combinatorial Exploration can bring, the succeeding sections display its corresponding breadth, offering samples of representations of the combinatorial objects and a few examples of strategies and resulting proof trees, sometimes derived by hand. We expect that experts in those combinatorial domains may have a better sense of effective object representations and decomposition strategies than we have given here, and the "plug-and-play" structure of our software enables researchers to easily implement new domains, new representations, and new strategies.

As discussed in Section 2.4, Combinatorial Exploration has been a successful tool for the enumeration of permutation classes. In this section, we will outline in more detail many of the particular strategies that have been used for this success.

6.1 Gridded permutations

Murphy and Vatter [121] introduced the notion of *grid classes* as a means of defining one permutation class as a geometric combination of others. We omit their definition here because we will use a new variation with a small but crucial change.

We have found that when working with geometric descriptions of sets of permutations, it is easier and vastly more effective to endow them with an additional geometric structure that we call a *gridding*. Extending this notion, we work with *gridded classes* instead of the grid classes of Murphy and Vatter (which from our viewpoint may be more appropriate to call *griddable* classes), and we will later define a combinatorial object called a *tiling* that implicitly defines a set of gridded permutations.

A *gridded permutation* of size n is a pair (π, P), where π is a permutation of length n called the *underlying permutation* and $P = (c_1, \ldots, c_n)$ for $c_i \in \mathbb{N} \times \mathbb{N}$ is the tuple of *positions*. The positions represent a placement of π onto the positive quadrant of $\mathbb{R}^2$ where the point corresponding to $(i, \pi(i))$ with $c_i = (x, y)$ has been drawn in the square $[x, x+1) \times [y, y+1)$. For example, Figure 10 depicts a gridded permutation of size 9.

Not every pair (π, P) is a valid gridded permutation, as the position tuple may not be consistent with the permutation. For example, $(21, ((0,0), (1,1)))$ is not a valid gridded permutation—there is no way to place one entry in the region $[0,1) \times [0,1)$ and another entry in the region $[1,2) \times [1,2)$ such that the two entries form a 21 pattern.

A pair (π, P) forms a valid gridded permutation when the relative positions of the entries in π are consistent with the relative positions of the cells in P. More formally, letting $|\pi| = n$ and $P = (c_1, \ldots, c_n)$ with $c_i = (x_i, y_i)$, the pair (π, P) forms a gridded permutation if $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_n$ and $y_{\pi^{-1}(1)} \leqslant y_{\pi^{-1}(2)} \leqslant \cdots \leqslant y_{\pi^{-1}(n)}$.

When all of the entries of a gridded permutation have the same position c, we abbreviate our notation by writing (π, c) instead of $(\pi, (c, c, \ldots, c))$. In this case we say the gridded permutation is *localized in cell c*.

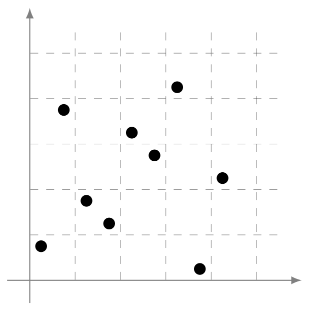


Figure 10: The gridded permutation $(284376915, ((0,0),(0,3),(1,1),(1,1),(2,3),(2,2),(3,4),(3,0),(4,2)))$. For example, the entry 9, which occurs at index 7 in the permutation, lies in the unit cell whose lower left corner has coordinates $(3,4)$, reflected by the fact that $(3,4)$ is the seventh cell in the tuple of cell positions.

Let $\mathcal{G}$ denote the set of all gridded permutations. The set $\mathcal{G}$ is inconvenient from a combinatorial perspective as, for example, it contains an infinite number of gridded permutations of size 1, and so we further define the refinements $\mathcal{G}^{(t,u)}$ to be the set of gridded permutations whose positions all lie within the rectangle $[0,t) \times [0,u)$. Informally, $\mathcal{G}^{(t,u)}$ is the set of those gridded permutations that can be drawn on a grid with width t (i.e., t columns) and height u (i.e., u rows). Let $\mathcal{G}_n^{(t,u)}$ denote the set of gridded permutations in $\mathcal{G}^{(t,u)}$ with size n. We can determine $|\mathcal{G}_n^{(t,u)}|$ by considering where the horizontal and vertical grid lines can pass between the points of each permutation of length n. One finds that the total number is

$$n!\binom{t+n-1}{t-1}\binom{u+n-1}{u-1}.$$

In particular, the finiteness of $|\mathcal{G}_n^{(t,u)}|$ implies that $\mathcal{G}^{(t,u)}$ is a combinatorial set.

The notion of pattern containment in permutations extends naturally to the realm of gridded permutations by requiring the entries of the smaller permutation to lie in precisely the same cells in the larger permutation. Formally, we say that the size n gridded permutation $g = (\pi, (c_1, \ldots, c_n))$ *contains* the size k gridded permutation $h = (\sigma, (d_1, \ldots, d_k))$ if there is a subsequence $\pi(i_1)\pi(i_2)\cdots\pi(i_k)$ of π whose standardization is equal to σ and $c_{i_j} = d_j$ for $1 \leqslant j \leqslant k$. Note that the cells of h are required to be actually equal to the cells of g where h occurs; there is no standardization of cells. We often use the term *pattern* colloquially to refer to the smaller permutations whose containment is under consideration. We call the subsequence $(i_1, \ldots, i_k)$ an *occurrence* of h in the gridded permutation g. If g does not contain h we say it *avoids* h. Figure 11 shows a gridded permutation that contains two occurrences of the pattern $(231, ((0,0),(1,1),(3,0)))$ and avoids the pattern $(231, ((1,3),(3,4),(4,2)))$.

Extending the notion of containment to sets of patterns, we say a gridded permutation g *avoids* a set of gridded patterns $\mathcal{O}$ if it avoids every gridded pattern in $\mathcal{O}$ and we define $\mathrm{Av}(\mathcal{O})$ to be the set of gridded permutations that avoid $\mathcal{O}$. On the other hand, we say a gridded permutation *contains* a set of gridded patterns $\mathcal{R}$ if it does not avoid $\mathcal{R}$. Note that this means g contains $\mathcal{R}$ if it contains *at least one* pattern in $\mathcal{R}$; g need not contain all patterns in $\mathcal{R}$. The set of gridded permutations containing $\mathcal{R}$ is denoted $\mathrm{Co}(\mathcal{R})$.[11]

[11]This definition of containment may seem odd at first, but its purpose will become clear in the following sub-

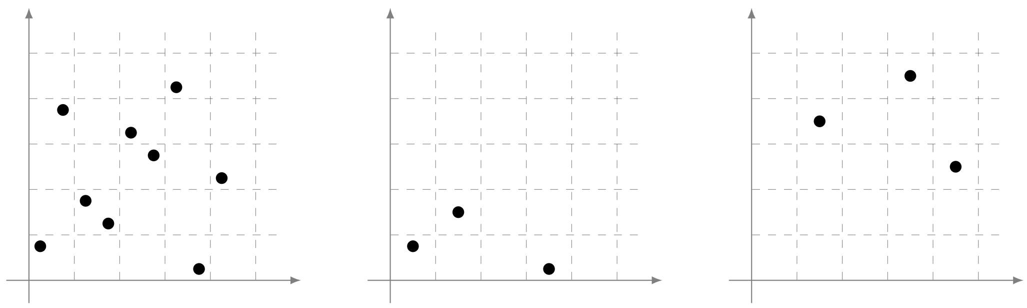


Figure 11: The gridded permutation from Figure 10 (left) contains two occurrences of the gridded pattern $(231, ((0,0),(1,1),(3,0)))$ (center), but avoids $(231, ((1,3),(3,4),(4,2)))$ (right).

6.2 Tilings

Rather than working with arbitrary combinatorial sets of gridded permutations, we restrict ourselves to those with a certain kind of structure that we now explain. In doing so, we are able to devise novel and efficient algorithms for manipulating these sets that exploit this structure. We start by defining a new combinatorial object that represents structured sets of gridded permutations.

Definition 6.1. A *tiling* is a triple $\mathcal{T} = ((t,u), \mathcal{O}, \mathcal{R})$ where t and u are integers, $\mathcal{O}$ is a set of gridded permutations that we call *obstructions* and $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_k\}$ is a set of sets of gridded permutations that we call *requirements*.

A tiling $\mathcal{T}$ represents the combinatorial set of gridded permutations $g \in \mathcal{G}^{(t,u)}$ such that g avoids $\mathcal{O}$ and g contains each $\mathcal{R}_i$ for $1 \leqslant i \leqslant k$. We call this set $\operatorname{Grid}(\mathcal{T})$. In other words, $\operatorname{Grid}(\mathcal{T})$ is the set gridded permutations in the region $[0,t) \times [0,u)$ that avoid *all* of the patterns in $\mathcal{O}$ and contain *at least one* of the patterns in each $\mathcal{R}_i$. We call the individual gridded permutations in each $\mathcal{R}_i$ requirements and we call each set $\mathcal{R}_i$ a *requirement list*.

An *interval* of a poset P is a set I with the property that for all $a, c \in I$ and $b \in P$, if $a \leqslant b \leqslant c$, then $b \in I$. The following proposition shows that the sets of gridded permutations that are represented by tilings are precisely intervals in the poset of gridded permutations when restricting to tilings of a fixed dimension. We see this as justification that tilings are a natural structure to use for Combinatorial Exploration.

Theorem 6.1. Fix positive integers t and u, and let $\mathcal{H}$ be a set of gridded permutations in $\mathcal{G}^{(t,u)}$. The following are equivalent:

1. The set $\mathcal{H}$ is an interval in the poset of gridded permutations $\mathcal{G}^{(t,u)}$.
2. There exists a tiling $\mathcal{T}$ such that $\operatorname{Grid}(\mathcal{T}) = \mathcal{H}$.

Proof. To show that Condition 1 implies Condition 2 we construct the tiling $\mathcal{T}$. Let $\mathcal{O}$ be the subset of gridded permutations in $\mathcal{G}^{(t,u)}$ that are not contained in any gridded permutation in $\mathcal{H}$. These will be the obstructions of $\mathcal{T}$; this set may be infinite, which is not a problem. The requirements will be a single requirement list containing every gridded permutation in $\mathcal{H}$, i.e.,

section, as will our use of the letters $\mathcal{O}$ and $\mathcal{R}$, which will come to represent what we call "obstructions" and "requirements" respectively.

$\mathcal{R} = \{\mathcal{H}\}$. Now define $\mathcal{T} = ((t, u), \mathcal{O}, \mathcal{R})$. It remains to show that $\mathrm{Grid}(\mathcal{T}) = \mathcal{H}$.

Let $g \in \mathrm{Grid}(\mathcal{T})$. By the definition of $\mathcal{T}$, g contains at least one permutation in $\mathcal{H}$ and avoids every obstruction in $\mathcal{O}$. Assume toward a contradiction that $g \notin \mathcal{H}$. Because $\mathcal{H}$ is an interval, g is not contained in any permutation in $\mathcal{H}$. Thus by construction, $g \in \mathcal{O}$. This is a contradiction thus g must be in $\mathcal{H}$. Conversely, consider a permutation $g \in \mathcal{H}$. This satisfies the requirements of $\mathcal{T}$ by the definition of $\mathcal{R} = \{\mathcal{H}\}$. Moreover, $\mathcal{O}$ was defined to be the set of permutations that are not contained in any permutation in $\mathcal{H}$, and so g does not contain any permutation in $\mathcal{O}$. Thus $g \in \mathrm{Grid}(\mathcal{T})$, confirming that Condition 1 implies Condition 2.

To show that Condition 2 implies Condition 1, let $\mathcal{T} = ((t, u), \mathcal{O}, \mathcal{R})$ be a tiling and define $\mathcal{H} = \mathrm{Grid}(\mathcal{T})$. With the aim of showing that $\mathcal{H}$ is an interval, suppose that $a, c \in \mathcal{H}$, $b \in \mathcal{G}^{(t,u)}$ and $a \leqslant b \leqslant c$. Since $c \in \mathcal{H}$, we know that c does not contain any obstruction in $\mathcal{O}$, and as $b \leqslant c$ the same must be true for b. Moreover a must contain at least one gridded permutation from each requirement list in $\mathcal{R}$, and as $b \geqslant a$ the same must be true for b. Since b avoids all of the obstructions of $\mathcal{T}$ and satisfies all of the requirements, we conclude $b \in \mathrm{Grid}(\mathcal{T}) = \mathcal{H}$ and therefore $\mathcal{H}$ is an interval. □

Henceforth, all tilings used in this paper have the property that each requirement list $\mathcal{R}_i$ contains only finitely many gridded permutations. Under this assumption, one useful property of the tiling representation is that for a tiling $\mathcal{T}$, it can be determined in finite time whether $\mathrm{Grid}(\mathcal{T})$ is empty. This is algorithmically useful, as later parts of this section will demonstrate.

Theorem 6.2. Consider a tiling $\mathcal{T} = ((t, u), \mathcal{O}, \mathcal{R})$ where $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$. Let ℓ_i be the size of the largest gridded permutation in R_i and define $L = \ell_1 + \cdots + \ell_k$. If $\mathrm{Grid}(\mathcal{T})$ is nonempty, then it contains a gridded permutation whose size is at most L.

Proof. Suppose $\mathrm{Grid}(\mathcal{T})$ is nonempty and let $\pi \in \mathrm{Grid}(\mathcal{T})$. Since π satisfies all of the requirements in $\mathcal{T}$, there exist gridded permutations $\sigma_1 \in \mathcal{R}_1, \ldots, \sigma_k \in \mathcal{R}_k$ such that π contains each σ_i. Choose one particular occurrence of each σ_i, let I_i be the indices of π where this occurrence is located, and let $I = I_1 \cup \cdots \cup I_k$. By design $|I| \leqslant L$.

Consider the subpermutation τ of π formed just from the entries at the indices of I. The permutation τ avoids all of the obstructions in $\mathcal{O}$ and satisfies all of the requirements in $\mathcal{R}$, and therefore $\tau \in \mathrm{Grid}(\mathcal{T})$ and has size at most L. □

This theorem implies that to check whether $\mathrm{Grid}(\mathcal{T})$ is empty, one only needs to check whether each gridded permutation in $\mathcal{G}^{(t,u)}$ of size at most L avoids the obstructions and contains the requirements. While this can be done in finite time, it is still slower than desirable. In practice, we have made several significant optimizations to speed up this algorithm, but we will not go into detail here.

Given any (ungridded) permutation class $\mathrm{Av}(B)$, define the tiling $\mathcal{T}_{\mathrm{Av}(B)}$ to be the 1×1 tiling with obstructions mimicking the basis elements B and no requirements:

$$\mathcal{T}_{\mathrm{Av}(B)} = ((1, 1), \{(\beta, (0, 0)) : \beta \in B\}, \{\}).$$

Since the map $\pi \mapsto (\pi, (0, 0))$ from $\mathrm{Av}(B)$ to $\mathrm{Grid}(\mathcal{T}_{\mathrm{Av}(B)})$ is a size-preserving bijection, we can enumerate $\mathrm{Av}(B)$ by applying Combinatorial Exploration to the set $\mathrm{Grid}(\mathcal{T}_{\mathrm{Av}(B)})$. The strategies we describe in the remainder of this section are designed to be applied to sets of gridded

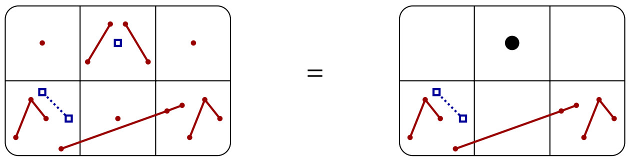


Figure 12: On the left, a tiling with all obstructions and requirements depicted. On the right, the same tiling shown using visual shortcuts.

permutations coming from tilings, that is, sets G such that $G = \mathrm{Grid}(\mathcal{T})$ for some tiling $\mathcal{T}$. The result is that we can define the decomposition function of a strategy as a function whose input is a tiling $\mathcal{T}$ and whose output is a sequence of tilings $(\mathcal{T}^{(1)}, \ldots, \mathcal{T}^{(m)})$, even though the decomposition function really acts on the corresponding sets of gridded permutations.

In order to explain the strategies we use for gridded permutations, we make liberal use of figures to depict tilings graphically. The entries of gridded permutations are drawn as points, and those points are connected by lines so it is clear which points belong to which gridded permutations. To maximize visual distinction, obstructions are drawn in red, with solid lines and solid round points, while requirements are drawn in blue, with dotted lines and hollow square points. Since the requirements of a tiling consist of a set of sets of gridded permutations $\{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$—a structure that is hard to convey visually—we typically only draw tilings when each $\mathcal{R}_i$ has size 1. Thus, in this article, whenever a tiling shows two requirements r_1 and r_2, for example, this should be interpreted as $\{\{r_1\}, \{r_2\}\}$ (two requirement lists of length one), not $\{\{r_1, r_2\}\}$ (one requirement list of length two), i.e., all gridded permutations that can be drawn on the tiling must contain both r_1 and r_2.

Consider, for example, the tiling $\mathcal{T} = ((3,2), \mathcal{O}, \mathcal{R})$ where[12]

$$\begin{aligned}\mathcal{O} = \{&(1,(0,1)), (1,(1,0)), (1,(2,1)), (12,(1,1)), (21,(1,1)),\\ &(132,(0,0)), (132,(2,0)), (123,((0,0),(2,0),(2,0)))\end{aligned}$$

and

$$\mathcal{R} = \{\{(1,(1,1))\}, \{(21,(0,0))\}\}$$

depicted on the left-hand side of Figure 12. The gridded permutations in $\mathrm{Grid}(\mathcal{T})$ are those contained in $\mathcal{G}^{(3,2)}$ that:

- have no entries in the cells $(0,1)$, $(1,0)$ and $(2,1)$,
- have exactly one entry in the cell $(1,1)$,
- have two entries in cell $(0,0)$ that form a 21 pattern,
- avoid the pattern 132 fully within cell $(0,0)$ or fully within cell $(2,0)$, and
- avoid a 123 pattern with the first entry in cell $(0,0)$ and the second two entries in cell $(2,0)$.

A cell is called *empty* if it contains the obstruction of length 1, and *nonempty* otherwise. It is so common to have cells that are empty or required to contain exactly one entry that we have

[12]Recall that when the entries of a gridded permutation all lie in the same cell c, we write (π, c) instead of $(\pi, (c, c, \ldots, c))$.

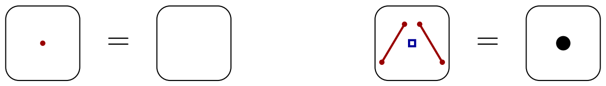


Figure 13: The two visual shortcuts that we use to make pictures of tilings more legible.

visual shortcuts for these two cases: cells that are required to be empty are drawn without the single-point obstruction, while cells required to contain exactly one entry are drawn with a solid, larger black circle. Figure 13 shows these two visual shortcuts. The tiling on the left-hand side of Figure 12 is thus simplified to the tiling on the right-hand side.

6.3 Descriptions of Six Strategies

To enumerate the (ungridded) permutation class $\mathrm{Av}(B)$, we start with the combinatorial set $\mathcal{T}_{\mathrm{Av}(B)}$ defined in the previous subsection and apply strategies that specifically decompose sets of gridded permutations of the form $\mathrm{Grid}(\mathcal{T})$ for a tiling $\mathcal{T}$.[13]

In this subsection we give intuitive, graphical descriptions of six fundamental strategies, saving the details and the proofs of their productivity for Subsection 6.5. The reader who chooses to skip Subsection 6.5 should still have a fairly complete understanding of these six strategies. There are many more strategies that can be efficaciously applied to the study of permutation patterns whose description we defer to a future work whose sole focus is on Combinatorial Exploration in the realm of permutation patterns.

In contrast to many algorithms that work by directly manipulating sets of permutations, like the enumeration schemes of Zeilberger [139] and the insertion encoding of Vatter [135], we can apply the strategies defined in this section by simply manipulating the tilings themselves, without the need to actually generate large sets of permutations. This is a major benefit of using tilings to represent sets of gridded permutations.

The six strategies we now introduce are, in order: requirement insertion, obstruction/requirement simplification, point placement, row and column separation, factorization, and obstruction inferral. The second, third, fourth, and sixth of these strategies are equivalence strategies; to prove their correctness in Subsection 6.5 we describe a size-preserving bijection between the input and output combinatorial sets. The first and fifth of these strategies are not equivalence strategies and we prove their productivity as outlined in Section 4.

We will consider the enumeration of $\mathcal{C} = \mathrm{Av}(1243, 1342, 2143)$ as a running example. A full proof tree for $\mathcal{C}$ is shown in Figure 24 on page 57. The root of the tree is $\mathcal{T}_{\mathcal{C}} = \mathcal{T}_1$ and each strategy used in the tree will be described in the remainder of this section.

6.3.1 Requirement Insertion

The strategy "size-0-or-not" in Example 3.2 used the fact that every gridded permutation either has size 0, or not, to decompose a combinatorial set $\mathcal{A}$ into a pair $d_Z(\mathcal{A}) = (\mathcal{B}, \mathcal{C})$ where $\mathcal{B}$ contains those elements with size 0 and $\mathcal{C}$ contains all others. This can be reformulated as

[13] As a result, all of our strategies implicitly have decomposition functions that output DNA when the input set is not a combinatorial set of this form.

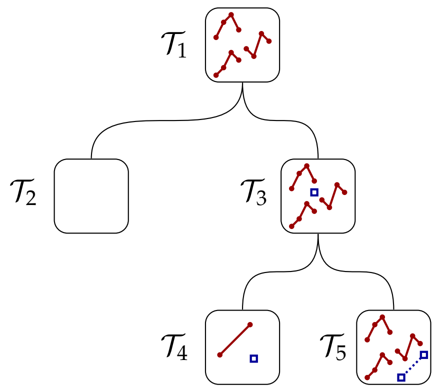


Figure 14: The top portion of the proof tree for $\mathrm{Av}(1243, 1342, 2143)$ shown in full in Figure 24 on page 57. The requirement $(1, (0, 0))$ is inserted into $\mathcal{T}_1$, and the requirement $(12, (0, 0))$ is inserted into $\mathcal{T}_3$. Note that the point requirement from $\mathcal{T}_3$ has not been shown on $\mathcal{T}_5$ because it is redundant.

follows: every gridded permutation π either contains a gridded permutation of size 1 (and thus $|\pi| \geqslant 1$) or avoids all gridded permutations of size 1 (and thus has size 0).

This idea can be easily generalized from containing or avoiding gridded permutations of size 1 to containing or avoiding any set H of gridded permutations. Two applications of this strategy are seen in Figure 14, which shows the top portion of the full proof tree in Figure 24 that is serving as our running example. We start with the tiling $\mathcal{T}_1$ and decompose it into the disjoint union of the tiling $\mathcal{T}_2$, which avoids the gridded pattern $(1, (0, 0))$, and the tiling $\mathcal{T}_3$ which contains this gridded pattern. The same strategy is then applied to $\mathcal{T}_3$ with the gridded pattern $(12, (0, 0))$. Note that $\mathcal{T}_5$ shows only the requirement $(12, (0, 0))$, and not the previous requirement $(1, (0, 0))$. This is because the new larger requirement makes the smaller requirement redundant, and so we have quietly removed it. We discuss this strategy in more detail in Subsection 6.3.2.

The tilings $\mathcal{T}_2$ and $\mathcal{T}_4$ are subject to verification strategies that we discuss in Subsection 6.4. That leaves the tiling $\mathcal{T}_5$, containing those permutations in $\mathcal{T}_1$ that are not strictly decreasing, as the only unexplored tiling so far.

More formally, let H be a set of gridded patterns and $\mathcal{T} = ((t, u), \mathcal{O}, \mathcal{R})$ a tiling. We define two tilings

$$\mathrm{ins}_{\mathcal{O}}(H, \mathcal{T}) = ((t, u), \mathcal{O} \cup H, \mathcal{R}), \qquad \mathrm{ins}_{\mathcal{R}}(H, \mathcal{T}) = ((t, u), \mathcal{O}, \mathcal{R} \cup \{H\}).$$

In the first one we have added all the gridded patterns in H as obstructions, and in the second one we have added H as one of the requirement lists, so every permutation in $\mathrm{Grid}(\mathrm{ins}_{\mathcal{R}}(H, \mathcal{T}))$ contains at least one gridded pattern in H. Clearly $\mathrm{Grid}(\mathcal{T})$ is equal to the disjoint union $\mathrm{Grid}(\mathrm{ins}_{\mathcal{O}}(H, \mathcal{T})) \sqcup \mathrm{Grid}(\mathrm{ins}_{\mathcal{R}}(H, \mathcal{T}))$. When $H = \{h\}$ we drop the brackets and instead write $\mathrm{ins}_{\mathcal{O}}(h, \mathcal{T})$ and $\mathrm{ins}_{\mathcal{R}}(h, \mathcal{T})$.

We now give the actual definition of the strategy. Given any set of gridded permutations H, the strategy $\texttt{ReqIns}_H$ is defined as follows[14]:

[14]Recall that we describe strategies by their action on tilings even though they actually act on combinatorial sets

– If $\mathcal{T}$ is a tiling with dimensions $t \times u$ and $H \subseteq \mathcal{G}^{(t,u)}$, and both sets

$$\text{ins}_{\mathcal{O}}(H,\mathcal{T}) \qquad \text{and} \qquad \text{ins}_{\mathcal{R}}(H,\mathcal{T})$$

are nonempty, then $d_{\texttt{ReqIns}_H}(\mathcal{T}) = (\text{ins}_{\mathcal{O}}(H,\mathcal{T}), \text{ins}_{\mathcal{R}}(H,\mathcal{T}))$. Otherwise $d_{\texttt{ReqIns}_H}(\mathcal{T}) =$ DNA.

– The reliance profile function is $r_{\texttt{ReqIns}_H}(n) = (n,n)$.

– The counting functions are $c_{\texttt{ReqIns}_H,(n)}((a_0,\ldots,a_n),(b_0,\ldots,b_n)) = a_n + b_n$.

Requirement Insertion is an example of what we call a *disjoint-union-type strategy*, which is a strategy S such that whenever $d_S(\mathcal{A}) = (\mathcal{B}^{(1)},\ldots,\mathcal{B}^{(m)})$ for $m > 1$ and all $\mathcal{B}^{(i)}$ are nonempty, we have that $\mathcal{A}$ is the disjoint union $\mathcal{B}^{(1)} \sqcup \cdots \sqcup \mathcal{B}^{(m)}$ and so $|\mathcal{A}_n| = |\mathcal{B}_n^{(1)}| + \cdots + |\mathcal{B}_n^{(m)}|$. We additionally require that the reliance profile function is $n \mapsto (n,\ldots,n)$. Under these conditions, disjoint-union-type strategies are easily seen to be productive; we justify the productivity of Requirement Insertion in particular in Theorem 6.3 in Subsection 6.5.

6.3.2 Obstruction and Requirement Simplification

In Figure 14, $\mathcal{T}_5$ is formed from $\mathcal{T}_3$ by adding the requirement list $\{(12,(0,0))\}$, so that the full requirements of $\mathcal{T}_5$ are

$$\{\ \{(1,(0,0))\},\ \ \{(12,(0,0))\}\ \}.$$

We have not drawn the requirement of size 1 in the figure because it is redundant—all gridded permutations that contain $(12,(0,0)))$ also contain $(1,(0,0))$—and so it is omitted from the figure. Throughout this work, we will frequently simplify obstructions and requirements in the way described here, often without mentioning it.

It is advantageous to remove redundant obstructions and requirements from tilings. In addition to a gain in computational efficiency, it also boosts the theoretical strength of Combinatorial Exploration by increasing our ability to identify when two tilings represent the same set of gridded permutations. There are several ways in which obstructions and requirements can be deleted, altered, or even added, leading to simpler tiling representations for the same sets of gridded permutations.

We want to first point out a subtlety around the way we have framed requirements. As we described in the example above, the requirement $(1,(0,0))$ is redundant in the sense that it is a strictly weaker condition than the requirement $(12,(0,0))$. It is incorrect to remove this redundancy just by deleting the requirement from its list, yielding the full requirements

$$\{\ \{\ \},\ \ \{(12,(0,0))\}\ \}.$$

A gridded permutation can be drawn on a tiling if it contains at least one requirement from each set, and so if a list becomes empty this condition is impossible to satisfy, implying that no gridded permutations can be drawn. The correct way to handle this redundancy is to instead delete the entire requirement list:

$$\{\ \{(12,(0,0))\}\ \}.$$

To summarize, deleting individual requirements out of their lists ostensibly leads to stricter conditions, while deleting an entire requirement list leads to a weaker condition.

of gridded permutations.

Obstruction Deletion
The case of obstructions is more straightforward, and so we start by defining a strategy that removes an obstruction if its removal does not change the underlying set of gridded permutations. We call this strategy Obstruction Deletion. Formally, for any gridded permutation h, define the equivalence strategy $\texttt{ObsDel}_h$ as follows:

- If $\mathcal{T} = ((t,u),\mathcal{O},\mathcal{R})$ is a tiling and if $h \in \mathcal{O}$, then define $\mathcal{T}' = ((t,u),\mathcal{O} \setminus \{h\},\mathcal{R})$. If $\text{Grid}(\mathcal{T}) = \text{Grid}(\mathcal{T}')$, then we define $d_{\texttt{ObsDel}_h}(\mathcal{T}) = \mathcal{T}'$. Otherwise $d_{\texttt{ObsDel}_h}(\mathcal{T}) = \texttt{DNA}$.
- The reliance profile function is $r_{\texttt{ObsDel}_h}(n) = (n)$.
- The counting functions are $c_{\texttt{ObsDel}_h,(n)}((a_0,\ldots,a_n)) = a_n$.

We should confess here that it may seem strange that we have defined a strategy that, by its very definition, makes absolutely no change to the actual combinatorial set under consideration—the more typical situation is that an equivalence strategy outputs a set different from its input, but that the two sets are equinumerous. Indeed, since this strategy only alters the representation of the set (the tiling), and not the set itself, we could have just described this as a computational step to simplify our representation completely independent of the strategic framework. However, there are cases where there are multiple tilings that could be used to describe the same set of gridded permutations, and (1) it is not clear whether one should be considered "simpler" than the other and (2) it may be algorithmically expensive to detect this. By writing such transformations as combinatorial rules, this information is preserved in our universe of rules and any version of the tiling can be used to construct a combinatorial specification.

Since $\texttt{ObsDel}_h$ is claimed to be an equivalence strategy, we do not need to prove that it is productive. In order to justify that it is an equivalence strategy, we would only need to verify that $|\text{Grid}_n(\mathcal{T})| = |\text{Grid}_n(\mathcal{T}')|$, but this is true by definition; we only apply the strategy in cases where the underlying gridded permutations do not change.

What this theoretical definition does not even begin to make clear is how, in the process of Combinatorial Exploration, we detect when it can be applied. It is, perhaps surprisingly, not always obvious when an obstruction can be deleted. (This is due to the complication added by the notion of requirements.) What we describe here is one sufficient condition that guarantees an obstruction can be deleted—when we detect this condition, we apply the strategy.

Suppose a tiling $\mathcal{T}$ has obstructions $h_1, h_2 \in \mathcal{O}$ with $h_1 \leqslant h_2$. Every gridded permutation that avoids h_1 also avoids h_2. Therefore, to any tiling $\mathcal{T}$ we may apply the strategy $\texttt{ObsDel}_h$ for any non-minimal $h \in \mathcal{O}$.

Requirement Deletion
There is a similar circumstance in which requirements can be deleted without altering the underlying set of gridded permutations. As before, we first define the strategy $\texttt{ReqDel}$, which gives no sense of in which situations it may be applied, and then we give a sufficient condition for a valid application of the strategy.

- If $\mathcal{T} = ((t,u),\mathcal{O},\mathcal{R})$ is a tiling and if $r \in \mathcal{R}_i$, then define
$$\mathcal{T}' = ((t,u),\mathcal{O},\{\mathcal{R}_1,\ldots,\mathcal{R}_{i-1},\mathcal{R}_i \setminus \{r\},\mathcal{R}_{i+1},\ldots,\mathcal{R}_k\}).$$
If $\text{Grid}(\mathcal{T}) = \text{Grid}(\mathcal{T}')$, then we define $d_{\texttt{ReqDel}_{r,i}}(\mathcal{T}) = \mathcal{T}'$. Otherwise $d_{\texttt{ReqDel}_{r,i}}(\mathcal{T}) = \texttt{DNA}$.

  - The reliance profile function is $r_{\texttt{ReqDel}_{r,i}}(n) = (n)$.
  - The counting functions are $c_{\texttt{ReqDel}_{r,i},(n)}((a_0, \ldots, a_n)) = a_n$.

As with `ObsDel`, `ReqDel` is an equivalence strategy that does not change the underlying set of gridded permutations. Suppose that a requirement list $\mathcal{R}_i$ contains two gridded permutations r_1 and r_2 with $r_1 \leqslant r_2$. Then, any gridded permutation that contains r_2 also contains r_1, and so r_2 can be deleted from $\mathcal{R}_i$ without changing the underlying gridded permutations. Therefore, a condition sufficient to ensure that $\texttt{ReqDel}_{r,i}$ can be applied to $\mathcal{T}$ is that $r \in \mathcal{R}_i$ is non-minimal among $\mathcal{R}_i$.

Requirement List Deletion
It is sometimes possible that an entire requirement list can be deleted without changing the underlying set of gridded permutations. Once again, we start by formally defining the strategy `ReqListDel`:

- If $\mathcal{T} = ((t, u), \mathcal{O}, \mathcal{R})$ is a tiling and $1 \leqslant i \leqslant |\mathcal{R}|$ then define

$$\mathcal{T}' = ((t, u), \mathcal{O}, \{\mathcal{R}_1, \ldots, \mathcal{R}_{i-1}, \mathcal{R}_{i+1}, \ldots, \mathcal{R}_k\}).$$

  If $\mathrm{Grid}(\mathcal{T}) = \mathrm{Grid}(\mathcal{T}')$, then we define $d_{\texttt{ReqListDel}_i}(\mathcal{T}) = \mathcal{T}'$. Otherwise $d_{\texttt{ReqListDel}_i}(\mathcal{T}) =$ `DNA`.
- The reliance profile function is $r_{\texttt{ReqListDel}_i}(n) = (n)$.
- The counting functions are $c_{\texttt{ReqListDel}_i,(n)}((a_0, \ldots, a_n)) = a_n$.

`ReqListDel` is an equivalence strategy that can be applied in the following scenario. Suppose $\mathcal{R}_i \in \mathcal{R}$ and there exists $j \neq i$ such that every $r' \in \mathcal{R}_j$ contains at least one $r \in \mathcal{R}_i$. Then, every gridded permutation that contains a requirement in the list $\mathcal{R}_j$ also contains a requirement in the list $\mathcal{R}_i$, and so $\mathcal{R}_i$ can be deleted without changing the underlying set of gridded permutations.

In each of the descriptions of `ObsDel`, `ReqDel`, and `ReqListDel`, we gave a sufficient but not necessary condition ensuring that each strategy could be applied. We discuss more general approaches in Subsection 6.3.6.

In the figures in this section that show examples of the application of strategies, we will often implicitly apply these obstruction and requirement simplification strategies in order to make the pictures more readable.

6.3.3 Point Placement

In describing the combinatorial specification for Av(132) in Section 3, we made the observation that the topmost point of any 132-avoiding permutation could be isolated, and we then made several inferences about the structure of such permutations. The strategy described in this subsection, point placement, is a vast generalization of this concept.

Point placement is an equivalence strategy that acts on a tiling by isolating one point of a singleton requirement in a cell of its own and forcing that point to be extreme in one of four directions: topmost, bottommost, leftmost, or rightmost. Before a more detailed definition, consider the example shown in Figure 15 which starts with a 1×1 tiling containing a 132 obstruction and a

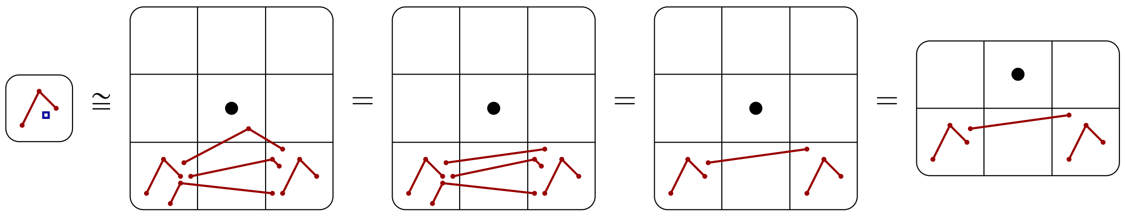


Figure 15: The evolution of tilings that result from performing the Point Placement strategy on the leftmost tiling. The first and second tiling are equivalent in the sense that there is a size-preserving bijection between the sets of gridded permutations associated with each. The third, fourth, and fifth tilings arise after applying the discussed simplifications, and are all equal to the second tiling in the sense that the sets of gridded permutations associated with each are all equal.

1 requirement. The single point of the requirement could be placed in any of the four extreme directions; here we place it topmost. The result is a 3×3 tiling in which the point of the requirement is in the middle cell. To ensure that this point has been placed in its own row and column, we add size 1 obstructions in cells $(1,0)$, $(0,1)$, $(2,1)$, and $(1,2)$. To ensure that the placed point is truly the topmost point, we add size 1 obstructions in cells $(0,2)$ and $(2,2)$. Lastly, to be sure that the gridded permutations on this new tiling still avoid 132, we add obstructions with the underlying 132 in all possible ways[15]. This second tiling can be simplified to the third tiling using a useful form of obstruction simplification that we elaborate on in Subsection 6.3.6: any gridded permutation that can be drawn on this tiling contains a point in the cell $(1,1)$, and thus avoiding the gridded permutation $(132, ((0,0),(1,1),(2,0)))$ is equivalent to avoiding the gridded subpermutation $(12, ((0,0),(2,0)))$. We can thus replace the former with the latter. The fourth tiling is obtained using the Obstruction Deletion strategy to remove those obstructions that contain $(12, ((0,0),(2,0)))$, and the fifth and final tiling is the result of deleting the topmost row which cannot contain any points of any gridded permutations anyway.

We took the time to point out each of these simplification steps individually because in future examples they will always be applied, often without comment, because otherwise the tilings produced by point placement have so many obstructions that pictures of them become useless. Applications of point placement can be more complicated than the example above in three ways: first, we may be placing a point of a requirement that has size greater than 1; second, there may be other requirement lists that have to be duplicated across new cells in a manner similar to obstructions; and third, we often apply point placement to tilings whose dimensions are larger than 1×1.

More formally, consider a tiling $\mathcal{T} = ((t,u), \mathcal{O}, \mathcal{R})$.

- If $\mathcal{T}$ contains a singleton requirement list $\mathcal{R}_i = \{h\}$, if ℓ is an index with $1 \leqslant \ell \leqslant |h|$, and $d \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$ is a direction, then we define

$$d_{\texttt{PointPl}_{h,\ell,d}}(\mathcal{T}) = \mathcal{T}',$$

where $\mathcal{T}'$ is formed as described in the example above, a process fully explained in Subsection 6.5.3. Otherwise $d_{\texttt{PointPl}_{h,\ell,d}}(\mathcal{T}) = \texttt{DNA}$.

[15]Many are redundant for obvious reasons, so we have chosen for clarity to not draw these.

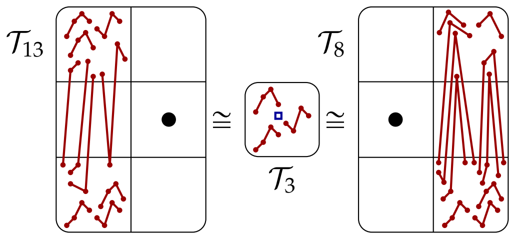


Figure 16: The tiling $\mathcal{T}_3$ is equivalent to each of the tilings $\mathcal{T}_8$ and $\mathcal{T}_{13}$, which are both produced by applying the point placement strategy to the same requirement in $\mathcal{T}_3$, but with different directions.

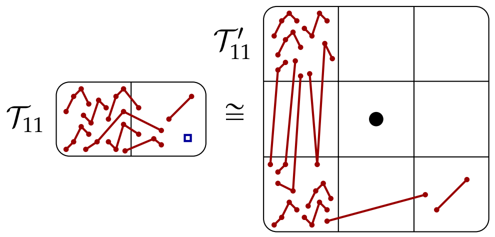


Figure 17: The tiling $\mathcal{T}'_{11}$ is the result of applying point placement to $\mathcal{T}_{11}$.

– The reliance profile function is $r_{\texttt{PointPl}_{h,\ell,d}}(n) = (n)$.

– The counting functions are $c_{\texttt{PointPl}_{h,\ell,d},(n)}((a_0,\ldots,a_n)) = a_n$.

The proof tree in Figure 24 uses point placement several times, so we take this opportunity to present these applications as further examples.

In Figure 16, the strategy $\texttt{PointPl}_{(1,(0,0)),1,\leftarrow}$ is applied to the tiling $\mathcal{T}_3$ to produce $\mathcal{T}_8$, while the strategy $\texttt{PointPl}_{(1,(0,0)),1,\rightarrow}$ is applied to the same tiling to produce $\mathcal{T}_{13}$.

For an example in which the input tiling has dimensions larger than 1×1, the tiling $\mathcal{T}'_{11}$ shown in Figure 17 is the result of applying $\texttt{PointPl}_{(1,(1,0)),1,\leftarrow}$ to T_{11}.

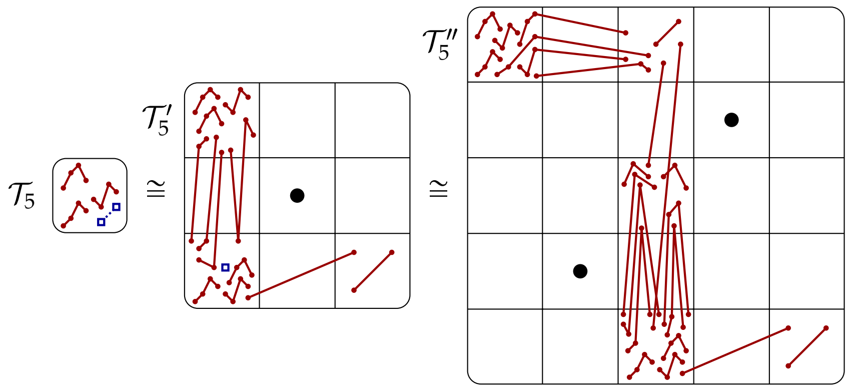


Figure 18: The tiling $\mathcal{T}''_5$ is the result of applying point placement twice to $\mathcal{T}_5$.

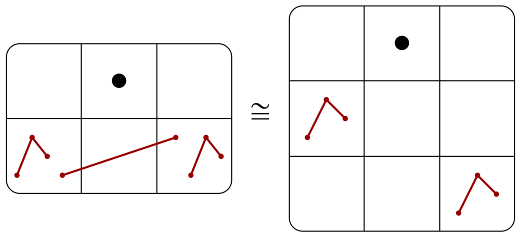


Figure 19: An application of row separation.

Finally, as demonstrated in Figure 18 the tiling $\mathcal{T}_5''$ is produced by twice applying point placement to $\mathcal{T}_5$, placing both points of a size two requirement. First, $\texttt{PointPl}_{(12,(0,0)),2,\rightarrow}$ is applied, placing the 2 in the requirement $(12,(0,0))$ as far to the right as possible. To the result, $\texttt{PointPl}_{(1,(0,0)),1,\leftarrow}$ is applied, placing the sole point of the size 1 requirement as far to the left as possible.

To prove that point placement is an equivalence strategy, we are required to show the existence of a size-preserving bijection between any input tiling and its corresponding output tiling. This requires rather a lot of bookkeeping, and is verified in Subsection 6.5.3.

6.3.4 Row Separation and Column Separation

The rightmost tiling in Figure 15 is the result of placing a point into a 1×1 tiling and performing several simplifications. The strategies described in this subsection, row separation and column separation, permit even further simplification of this tiling and others.

The presence of the $(12,(0,0),(2,0))$ obstruction in that tiling implies that for any gridded permutation drawn on the tiling, any entries in cell $(0,0)$ must lie above any entries in cell $(2,0)$. To capture this information, we create a new tiling, shown on the right in Figure 19, in which the content of these two cells has been separated into two rows.

Another example of this strategy is found in the tiling $\mathcal{T}_5''$ (which itself was the result of twice applying point placement to $\mathcal{T}_5$) in the proof tree in Figure 24. As we show in Figure 20, the obstruction $(12,((2,0),(4,0)))$ implies that the two nonempty cells in the bottom row can be separated. Further, the two obstructions $(12,((2,0),(2,4)))$ and $(12,((2,2),(2,4)))$ imply that any entries in cells $(2,0)$ and $(2,2)$ must lie to the right of any entries in cell $(2,4)$, and so the cells in this column can be separated, yielding the tiling $\mathcal{T}_6$ in Figure 24.

We now give the formal definition of the row separation equivalence strategy; the column separation strategy is defined similarly. Consider a tiling $\mathcal{T} = ((t,u),\mathcal{O},\mathcal{R})$. For a particular row r, let S be a nonempty subset of the nonempty cells in r and let S' denote the remaining nonempty cells in r. We can apply row separation to row r, splitting it in two rows in which the lower row inherits the cells from S and the upper row inherits the cells from S', if there is no gridded permutation that can be drawn on $\mathcal{T}$ that possesses an entry in a cell in S whose value is larger than an entry in a cell in S'. A sufficient condition to ensure this is the following: for every pair of cells $(c,c') \in S \times S'$, if c is to the left of c' then $\mathcal{O}$ contains the obstruction $(21,(c,c'))$, otherwise if c is to the right of c' then $\mathcal{O}$ contains the obstruction $(12,(c',c))$.

As the examples above suffice to understand the idea of row and column separation, our formal

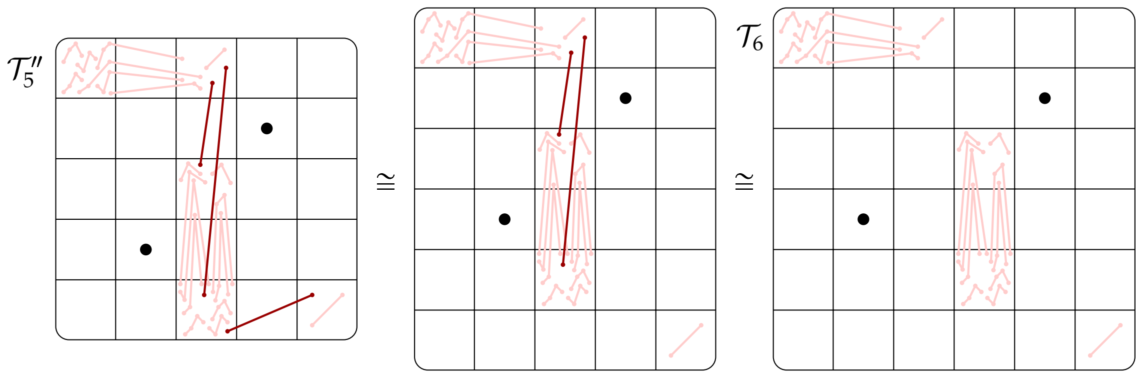


Figure 20: The tiling $\mathcal{T}_6$ is the result of applying row separation and then column separation to $\mathcal{T}_5''$.

definition below omits the full details, which are deferred to Subsection 6.5.4.

- For a tiling $\mathcal{T} = ((t,u), \mathcal{O}, \mathcal{R})$, a row r, and a nonempty subset S of the nonempty cells in row r, if the obstructions described above are all present, then we define

  $$d_{\texttt{RowSep}_{r,S}}(\mathcal{T}) = \mathcal{T}'$$

  where $\mathcal{T}'$ is the tiling in which the cells in S have been moved into a separate row below the cells in row r but not in S, and the obstructions and requirements have been adjusted accordingly. Otherwise $d_{\texttt{RowSep}_{r,S}}(\mathcal{T}) = \texttt{DNA}$.
- The reliance profile function is $r_{\texttt{RowSep}_{r,S}}(n) = (n)$.
- The counting functions are $c_{\texttt{RowSep}_{r,S}}((a_0, \ldots, a_n)) = a_n$.

The column separation strategy $\texttt{ColSep}_{c,S}$ is defined analogously.

6.3.5 Factor

Most of the strategies we have already described transform tilings into new tilings that are in some way more complicated—larger, more obstructions, more requirements, etc. Factorization is a strategy that identifies when pieces of a tiling do not interact with each other, and splits them into several subtilings. This tends to lead to combinatorial specifications that are recursive.

We say that two subsets S_1 and S_2 of cells of a tiling are *non-interacting* if no cell of S_1 shares a row or column with any cell of S_2 and if there is no obstruction or requirement list that involves cells in both S_1 and S_2. Figure 21 shows a 6×6 tiling that has 5 minimal pairwise non-interacting sets of cells:

$$\{(0,5),(2,5)\} \qquad \{(1,2)\} \qquad \{(3,1),(3,3)\} \qquad \{(4,4)\} \qquad \{(5,0)\},$$

although the factorization actually performed in that figure leaves the non-interacting sets $\{(1,2)\}$ and $\{(3,1),(3,3)\}$ together. Why do we not use the full factorization? This is the magic of Combinatorial Exploration—it has discovered that this partial factorization permits the discovery of the proof tree from Figure 24 that we are currently discussing, whereas a human searching for this proof tree by hand may not have gone down this path.

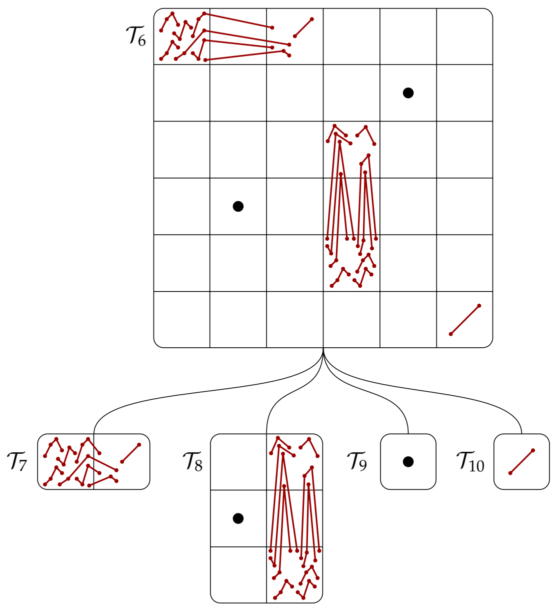


Figure 21: The tiling $\mathcal{T}_6$ factors into four tilings.

The counting functions of the factorization strategy are more interesting than the other strategies we have discussed. Suppose $\mathcal{A}$ is a tiling that factors into two tilings $\mathcal{B}^{(1)}$ and $\mathcal{B}^{(2)}$. The non-interactivity of the cells that became $\mathcal{B}^{(1)}$ with the cells that became $\mathcal{B}^{(2)}$ imply that each gridded permutation α of size n that can be drawn on $\mathcal{A}$ can be formed uniquely from a pair β_1, β_2 where β_1 can be drawn on $\mathcal{B}^{(1)}$, β_2 can be drawn $\mathcal{B}^{(2)}$, and $|\beta_1| + |\beta_2| = n$. Therefore,

$$|\mathcal{A}_n| = \sum_{i=0}^{n} |\mathcal{B}^{(1)}_i||\mathcal{B}^{(2)}_{n-i}|.$$

More generally, when a tiling $\mathcal{A}$ factors into m tilings $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$ we have

$$|\mathcal{A}_n| = \sum_{i_1+\cdots+i_m=n} |\mathcal{B}^{(1)}_{i_1}| \cdots |\mathcal{B}^{(m)}_{i_m}|.$$

However, for the first time, we are in danger of defining a strategy that does not satisfy the productivity conditions defined in Section 4. Consider, for example, the factorization shown in Figure 22 in which a tiling is factored into two subtilings, one containing just a point, and the other containing two cells. Given the general counting formula above, it feels natural to define the reliance profile function of a strategy that factors a tiling into two subtilings to be

$$r(n) = (n, n).$$

Figure 22 reveals a problem: the subtiling $\mathcal{B}^{(2)}$ has counting sequence $|\mathcal{B}^{(2)}_n| = 2^n$ while the original tiling has a termwise strictly smaller counting sequence $|\mathcal{A}_n| = 2^{n-1}$, violating Condition 2(a) of Definition 4.2 that requires

If $\mathcal{A}_N$ relies on $\mathcal{B}^{(i)}_N$ for some $N \in \mathbb{N}$, then $|\mathcal{A}_n| \geqslant |\mathcal{B}^{(i)}_n|$ for all $n \in \mathbb{N}$.

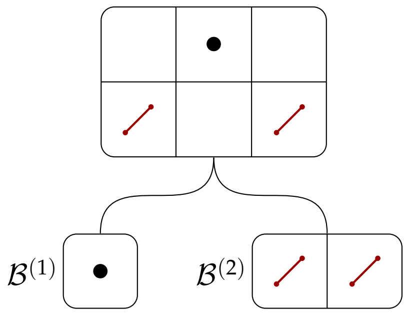


Figure 22: An example of factoring that demonstrates that productivity relies on careful definition of different reliance profile functions for different situations.

This problem is easily mitigated with a bit of care. In fact, the enumeration of $\mathcal{A}_n$ does not actually depend on the enumeration of $\mathcal{B}_n^{(2)}$ because $|\mathcal{B}_0^{(1)}| = 0$, simplifying the counting formula slightly by eliminating the $i = 0$ term in the summation:

$$|\mathcal{A}_n| = \sum_{i=0}^{n} |\mathcal{B}_i^{(1)}||\mathcal{B}_{n-i}^{(2)}| = \sum_{i=1}^{n} |\mathcal{B}_i^{(1)}||\mathcal{B}_{n-i}^{(2)}|.$$

More generally, we will define the factorization strategy so that the enumeration of $\mathcal{A}_n$ for the parent tiling $\mathcal{A}$ only depends on the enumeration of the size n gridded permutations in a child tiling $\mathcal{B}^{(i)}$ when absolutely required. We prove in Subsection 6.5.5 that the resulting strategy is always productive. According to this new definition, the reliance profile function for the factorization in Figure 22 is

$$r(n) = (n, n-1).$$

Formally, consider a tiling $\mathcal{T} = ((t, u), \mathcal{O}, \mathcal{R})$.

– Let P be a partition of the nonempty cells of $\mathcal{T}$ into m parts, and for concreteness consider the parts of P to be indexed in increasing order by their lexicographically smallest cell. If the cells of any part of P interact with the cells of any other part, then $\mathcal{T}$ cannot be factored according to this partition of cells. Thus, assume now that P is such that the parts are non-interacting, so that $\mathcal{T}$ will be factored into subtilings $\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}$. Assume that each of these subtilings contains at least one gridded permutation of size at least 1. Define

  $$S = \{i \in \{1, \ldots, m\} \,:\, |\mathcal{B}_0^{(j)}| = 0 \text{ for some } j \neq i\}.$$

  If $i \in S$, then the enumeration of $\mathcal{T}_n$ will *not* rely on the enumeration of $\mathcal{B}_n^{(i)}$ because $|\mathcal{B}_0^{(j)}| = 0$ for some $j \neq i$.

  With such P and S, we define

  $$d_{\mathtt{Factor}_{P,S}}(\mathcal{T}) = (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(m)}).$$

  In the case of an incompatible partition P or set S, or where some $\mathcal{B}^{(i)}$ does not contain a gridded permutation of size at least 1, we define

  $$d_{\mathtt{Factor}_{P,S}}(\mathcal{T}) = \mathtt{DNA},$$

as usual.

– Fix a partition P and its corresponding set S. The reliance profile function is

$$r_{\texttt{Factor}_{P,S}}(n) = (r^{(1)}(n), \dots, r^{(m)}(n))$$

where

$$r^{(i)}(n) = \begin{cases} n-1, & \text{if } i \in S \\ n, & \text{if } i \notin S \end{cases}.$$

– To describe the counting function we first define vectors of indeterminates

$$b^{(i)} = \begin{cases} (b_0^{(i)}, \dots, b_{n-1}^{(i)}), & \text{if } i \in S \\ (b_0^{(i)}, \dots, b_n^{(i)}), & \text{if } i \notin S \end{cases}.$$

The counting functions are

$$c_{\texttt{Factor}_{P,S},(n)}(b^{(1)}, \dots, b^{(m)}) = \sum_{(i_1,\dots,i_m) \in I} b_{i_1}^{(1)} \cdots b_{i_m}^{(m)},$$

where the sum is over

$$I = \{(i_1, \dots, i_m) \in \{0, \dots, n\}^m \, : \, i_1 + \cdots + i_m = n \text{ and } i_\ell \neq n \text{ if } \ell \in S\}.$$

To illustrate this slightly intensive definition, the strategy applied to the parent tiling in Figure 21 is $\texttt{Factor}_{P,S}$ where

$$P = \{ \quad \{(0,5),(2,5)\}, \quad \{(1,2),(3,1),(3,3)\}, \quad \{(4,4)\}, \quad \{(5,0)\} \quad \}$$

and $S = \{1, 2, 3, 4\}$. The reliance profile function is

$$r_{\texttt{Factor}_{P,S}}(n) = (n-1, n-1, n-1, n-1)$$

and the counting functions are

$$c_{\texttt{Factor}_{P,S},(n)}(b^{(1)}, b^{(2)}, b^{(3)}, b^{(4)}) = \sum_{\substack{i_1+i_2+i_3+i_4=n \\ 0 \leqslant i_1,i_2,i_3,i_4 \leqslant n-1}} b_{i_1}^{(1)} b_{i_2}^{(2)} b_{i_3}^{(3)} b_{i_4}^{(4)}.$$

6.3.6 Obstruction Inferral

It is sometimes possible to place an additional obstruction onto a tiling without changing the underlying set of gridded permutations. For a rather trivial example, see the tiling in Figure 23, in which the obstruction $(1, (1,0))$ can be added. Sometimes, but not always, this inferred obstruction is a subobstruction of an existing obstruction.

As with some of the strategies that we have previously introduced, we will start with a simple definition and then follow it with the more detailed discussion about when the strategy can be applied. Formally, the strategy $\texttt{ObsInf}_h$ is defined as follows.

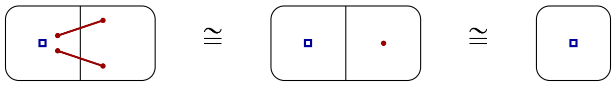


Figure 23: On the left, an example of a tiling to which the obstruction inferral strategy can be applied. In the middle, the result of applying obstruction inferral. On the right, an equivalent tiling that results from deleting the rightmost cell, which cannot contain any entries.

– If $\mathcal{T}$ is a tiling with dimensions $t \times u$, $h \in \mathcal{G}^{(t,u)} \setminus \mathcal{O}$, and

$$\text{Grid}(\mathcal{T}) = \text{Grid}(((t,u), \mathcal{O} \cup \{h\}, \mathcal{R})),$$

then $d_{\texttt{ObsInf}_h}(\mathcal{T}) = ((t,u), \mathcal{O} \cup \{h\}, \mathcal{R})$. Otherwise $d_{\texttt{ObsInf}_h}(\mathcal{T}) = \texttt{DNA}$.

– The reliance profile function is $r_{\texttt{ObsInf}_h}(n) = (n)$.

– The counting functions are $c_{\texttt{ObsInf}_h,(n)}((a_0, \ldots, a_n)) = a_n$.

Like many of our previous strategies, Obstruction Inferral does not actually change the set of gridded permutations at all, only altering the tiling representation used for the set. As such, it is true by definition that Obstruction Inferral is a valid equivalence strategy.

We will now discuss two ways to determine when Obstruction Inferral can be applied to a tiling, the first being more limited but computationally easy and the second being fully general but computationally intensive.

First Obstruction Inferral Case

In discussing the point placement strategy in Section 6.3.3, we mentioned in Figure 15 that the obstruction $(132, ((0,0), (1,1), (2,0)))$ could be replaced with the subobstruction $(12, ((0,0), (2,0)))$ because of the point cell $(1,1)$. This phenomenon can be detected more generally. Suppose h is an obstruction on the tiling $\mathcal{T}$ and partition the entries of h maximally such that no two entries in different parts occur in cells that share a row or column. Call this partition P. If there exists a part $p \in P$ such that the subgridded permutation formed by the entries in p is contained in every requirement in some requirement list, then the smaller obstruction formed by deleting the entries in p from h can be added to $\mathcal{T}$ without changing the underlying set of gridded permutations.

In the example from Figure 15, P has two parts, the first containing the subgridded permutation $(12, ((0,0), (2,0)))$ and the second containing the subgridded permutation $(1, ((1,1)))$. Since $(1,1)$ is a point cell, the subgridded permutation coming from this part of the partition can be deleted from the obstruction.

To see that this claim is true in general, suppose the tiling T has an obstruction h, and that in the partition P defined above there is part p such that the subobstruction h' arising from deleting the subgridded permutation corresponding to p from h can be inferred. Let R be the requirement list that enabled this inferral (i.e., every requirement in R contains as a pattern the subgridded permutation formed by the entries in p). We claim that no gridded permutation that can be drawn on $\mathcal{T}$ contains the pattern h'. To the contrary, suppose there were such a permutation π that contained h'. Since $\pi \in \text{Grid}(\mathcal{T})$, it must also contain at least one of the requirements in R. All of these requirements contain the pattern formed by the entries in p.

Crucially, since the cells involved in p share no rows nor columns with the cells involved in other parts of P, the only manner in which a gridded permutation can contain h' and R is by containing the full obstruction h.

Second Obstruction Inferral Case
Theorem 6.2 shows that it can be checked in finite time whether $\mathrm{Grid}(\mathcal{T}) = \varnothing$ for any given tiling $\mathcal{T}$. We can use this fact to determine precisely when an obstruction may be inferred, although the computational burden is significant.

To infer whether an obstruction h may be added onto the tiling $\mathcal{T} = ((t, u), \mathcal{O}, \mathcal{R}))$, we form a new tiling $\mathcal{T}'$ in which the potential new obstruction is instead placed in a new requirement list:

$$\mathcal{T}' = ((t, u), \mathcal{O}, \mathcal{R} \cup \{h\}).$$

If $\mathrm{Grid}(\mathcal{T}')$ is empty, we can conclude that no gridded permutation that can be drawn on $\mathcal{T}$ contains the pattern h. Therefore, h may be inferred as an obstruction on $\mathcal{T}$ without eliminating any of its gridded permutations.

6.4 Verification Strategies

In Subsection 3.3, we discussed the concept of "verification strategies" – nullary strategies that represent when the enumeration of a combinatorial set is known independently from the current Combinatorial Exploration process. The proof tree for $\mathrm{Av}(1243, 1342, 2143)$ in Figure 24 uses four different verification strategies, and so now that we have introduced several strategies in the previous subsection, we will take this opportunity to trace through the tree and see how they are applied, pointing out the verification strategies as we encounter them.

We want to emphasize that this proof tree is the *output* of successful Combinatorial Exploration. The process of finding this tree involves the discovery of many combinatorial rules, and sometime after all rules in this particular tree have been discovered, the algorithm notices (using Algorithm 1 in Subsection 3.5) that this particular subset of rules forms a combinatorial specification.

We previously discussed the top part of the tree in Figure 14. To the root tiling $\mathcal{T}_1$, we apply the requirement insertion strategy. Inserting $H_1 = \{(1, ((0, 0)))\}$ into $\mathcal{T}_1$ creates the rule $\mathcal{T}_1 \xleftarrow{\mathtt{ReqIns}_{H_1}} (\mathcal{T}_2, \mathcal{T}_3)$.

The tiling $\mathcal{T}_2$ represents the set containing only the empty gridded permutation of size 0. This is clearly a set whose enumeration is known *a priori* and so to it we apply the verification strategy $V_{\mathcal{T}_2}$ (recall from Section 3.3 that each set corresponds to a unique verification strategy). The result is a rule $\mathcal{T}_2 \xleftarrow{V_{\mathcal{T}_2}} ()$.

Moving further into the tree, the insertion of $H_2 = \{(12, ((0, 0), (0, 0)))\}$ into $\mathcal{T}_3$ gives the rule $\mathcal{T}_3 \xleftarrow{\mathtt{ReqIns}_{H_2}} (\mathcal{T}_4, \mathcal{T}_5)$.[16] The set of gridded permutations represented by $\mathcal{T}_4$ are precisely those single-celled permutations that are strictly decreasing and have size at least 1. The enumeration

[16]Recall from our previous discussion that technically $\mathcal{T}_5$ should have a point requirement, but it is detected to be redundant and removed. Although in practice this should be depicted as its own rule (an equivalence strategy), we have hidden it, and several other such simplification, from this figure.

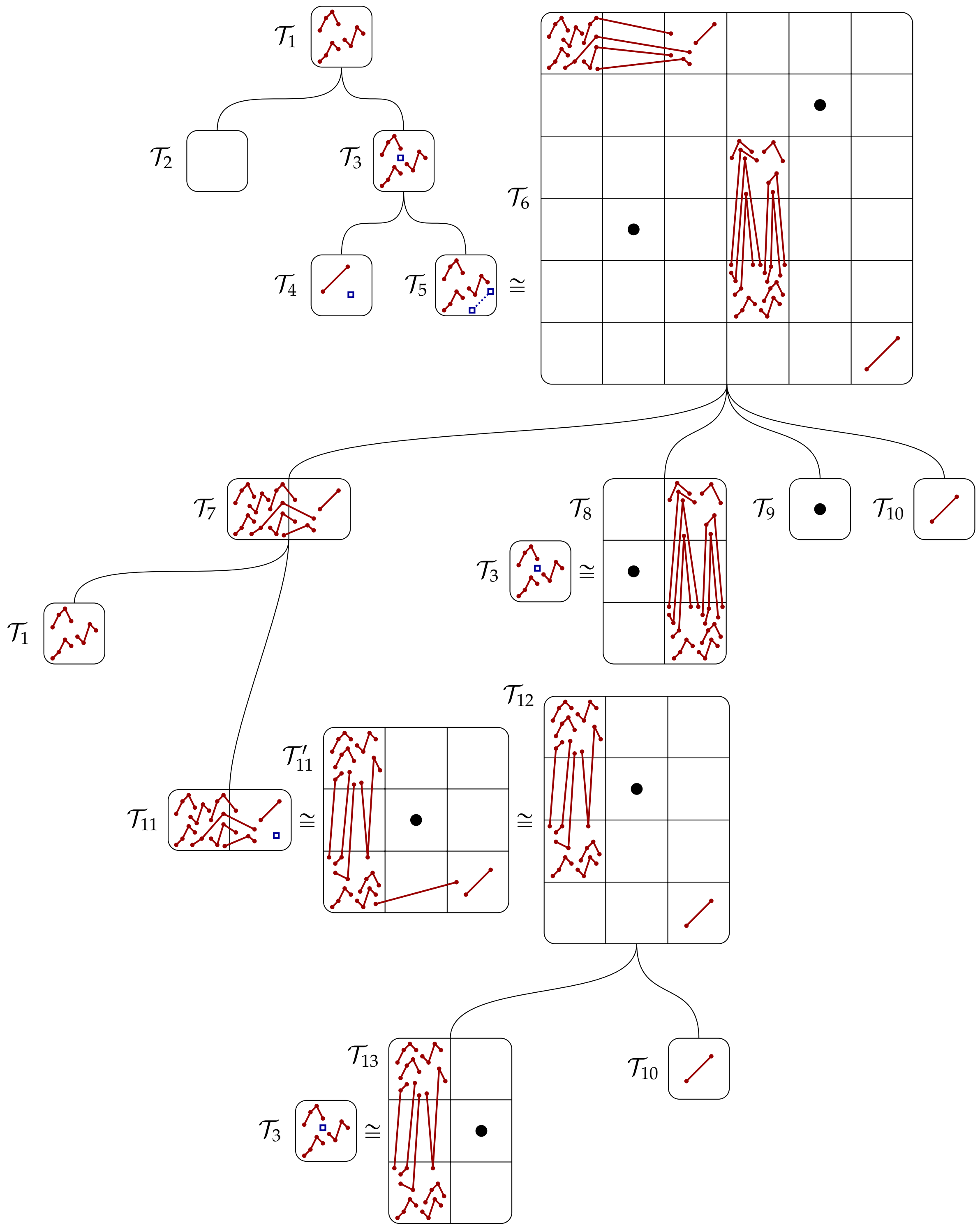


Figure 24: A pictorial representation of the combinatorial specification found by Combinatorial Exploration for Av(1243, 1342, 2143).

of these permutations is evidently $a_0 = 0$ and $a_n = 1$ for $n \geqslant 1$, and since this is known independently of the structural decomposition being described, we can apply the verification strategy $V_{\mathcal{T}_4}$ to produce the rule $\mathcal{T}_4 \xleftarrow{V_{\mathcal{T}_4}} ()$.

As this shows, a researcher who is applying Combinatorial Exploration has a lot of flexibility in choosing to which combinatorial sets we can apply a verification strategy. In this particular case, had we not employed the strategy $V_{\mathcal{T}_4}$ we could still have easily completed this part of the proof tree by applying the point placement strategy to the requirement in $\mathcal{T}_4$, then applying the factor strategy to the result.

It is imperative to verify the two *atomic sets*, the set containing only the empty permutation ($\mathcal{T}_2$ here) and the set containing only the gridded permutation of size 1 ($\mathcal{T}_9$ here). When deciding which other sets should be verified, one should keep in mind that the goal is to shorten the search for a proof tree by identifying sets that can be independently enumerated, and therefore eliminating the need to expand them further.

There are many domain-specific algorithms to enumerate certain sets of permutations in polynomial time. Here we will just briefly mention one: the insertion encoding of Vatter [135], can be extended to some single-row or single-column tilings. In these cases, one can be certain that a combinatorial set can be enumerated, and so for the sake of computational efficiency it makes sense to apply verification strategies to them.

For a slightly more experimental search, one might choose to verify combinatorial sets that they simply suspect could be enumerated if needed, either by hand or with a separate self-contained application of Combinatorial Exploration. We have found this to be effective in the domain of permutation patterns, where we often use the rule-of-thumb that when we are searching for a proof tree for a permutation class $\mathrm{Av}(B)$ then any 1×1 tiling whose combinatorial set is a subclass $\mathrm{Av}(B')$ can be verified (even if it carries requirements as in the case of $\mathcal{T}_4$ above) as long as B' is obtained from B by adding a pattern of length at most the length of the longest pattern in B. This heuristic is not guaranteed to work; for example, there is a permutation class that avoids two patterns of length 4 and has an algebraic generating function such that if you add a particular third pattern of length 4 to its basis, the resulting class has an unknown generating function that is conjectured to be non-D-finite [9].

Continuing the traversal in the proof tree for $\mathrm{Av}(1243, 1342, 2143)$, we are now at tiling $\mathcal{T}_5$. After applying two point placements as shown in Figure 18, followed by the row and column separations shown in Figure 20 we obtain a chain of equivalence rules between $\mathcal{T}_5$ and $\mathcal{T}_6$. Applying the factor strategy to $\mathcal{T}_6$ gives us four tilings, $\mathcal{T}_7$, $\mathcal{T}_8$, $\mathcal{T}_9$ and $\mathcal{T}_{10}$. The tiling $\mathcal{T}_9$ is verified as an atom, and the tiling $\mathcal{T}_{10}$ represents a subclass of the class being explored, and so as discussed above we mark it verified as well. The tiling $\mathcal{T}_8$ is already known to be equivalent to $\mathcal{T}_3$, because earlier in the process of Combinatorial Exploration, we applied point placement to $\mathcal{T}_3$ and obtained $\mathcal{T}_8$. The last tiling, $\mathcal{T}_7$, cannot be verified so we must continue to decompose it. So, we apply requirement insertion with $H = \{(1, ((1, 0)))\}$ (a point in cell $(1, 0)$) to it. When H is avoided, one obtains the left-hand child, which is identical to $\mathcal{T}_1$, the root. When H is contained, one obtains the right-hand child, $\mathcal{T}_{11}$, to which we apply point placement and row separation. This produces the tiling $\mathcal{T}_{12}$ that factors into $\mathcal{T}_{10}$ (which we have already seen and verified) and $\mathcal{T}_{13}$ which is equivalent to $\mathcal{T}_3$ via a point placement.

This set of combinatorial rules forms a specification, and since all of the rules are produced by productive strategies, we are guaranteed that the specification can be used to produce terms of the counting sequence for Av$(1243, 1342, 2143)$ in polynomial time.

Below, we show the full specification on the left (omitting the names of the strategies, which would typically be shown over the arrows, for space), and the corresponding system of generating functions on the right. As previously discussed, combinatorial sets that are equivalent are contracted into an equivalence class in the specification; in this case, this occurs three times. We let $\mathcal{E}_3$, $\mathcal{E}_5$, and $\mathcal{E}_{11}$ denote the equivalence classes $\{\mathcal{T}_3, \mathcal{T}_8, \mathcal{T}_{13}\}$, $\{\mathcal{T}_5, \mathcal{T}_6\}$, and $\{\mathcal{T}_{11}, \mathcal{T}'_{11}, \mathcal{T}_{12}\}$, respectively.

$$\begin{aligned}
\mathcal{T}_1 &\leftarrow (\mathcal{T}_2, \mathcal{E}_3) & T_1(x) &= T_2(x) + E_3(x)\\
\mathcal{T}_2 &\leftarrow () & T_2(x) &= 1\\
\mathcal{E}_3 &\leftarrow (\mathcal{T}_4, \mathcal{E}_5) & E_3(x) &= T_4(x) + E_5(x)\\
\mathcal{T}_4 &\leftarrow () & T_4(x) &= x/(1-x)\\
\mathcal{E}_5 &\leftarrow (\mathcal{T}_7, \mathcal{E}_3, \mathcal{T}_9, \mathcal{T}_{10}) & E_5(x) &= T_7(x) \cdot E_3(x) \cdot T_9(x) \cdot T_{10}(x)\\
\mathcal{T}_7 &\leftarrow (\mathcal{T}_1, \mathcal{E}_{11}) & T_7(x) &= T_1(x) + E_{11}(x)\\
\mathcal{T}_9 &\leftarrow () & T_9(x) &= x\\
\mathcal{T}_{10} &\leftarrow () & T_{10}(x) &= 1/(1-x)\\
\mathcal{E}_{11} &\leftarrow (\mathcal{E}_3, \mathcal{T}_{10}) & E_{11}(x) &= E_3(x) \cdot T_{10}(x)
\end{aligned}$$

The system of equations can be solved to find the generating function for the class:

$$T_1(x) = \frac{1 + x - \sqrt{1 - 6x + 5x^2}}{2x(2-x)}.$$

6.5 Further Details of the Six Presented Strategies

In this subsection, we will provide full details of each strategy described in Subsection 6.3, and prove the productivity of the relevant ones. Although each strategy can be easily understood on an intuitive level from our earlier rough descriptions, the full details and proofs often involve quite a bit of detailed argument.

6.5.1 Requirement Insertion

The Requirement Insertion strategy decomposes a set of gridded permutations into those that avoid all gridded permutations from a set H and those that contain at least one gridded permutation from H. The formal definition of Requirement Insertion is given in Subsection 6.3.1 on page 43. It is a disjoint-union-type strategy, and so the proof of its productivity is rather simple.

Theorem 6.3. The Requirement Insertion strategy is productive.

Proof. We need to verify Conditions 1 and 2 of Definition 4.2. The reliance profile function of this strategy is $n \mapsto (n, n)$, which ensures that Condition 1 is satisfied.

In order to check Condition 2, suppose that $\mathcal{A}$ is a set of gridded permutations, and let H be the set of patterns under consideration. Let $\mathcal{B}$ and $\mathcal{C}$ be the subsets of $\mathcal{A}$ that avoid or contain H, respectively. Condition 2 requires that:

(a) $|\mathcal{A}_n| \geqslant |\mathcal{B}_n|$ and $|\mathcal{A}_n| \geqslant |\mathcal{C}_n|$ for all $n \in \mathbb{N}$;

(b) $|\mathcal{A}_k| > |\mathcal{B}_k|$ for some $k \in \mathbb{N}$ and $|\mathcal{A}_\ell| > |\mathcal{C}_\ell|$ for some $\ell \in \mathbb{N}$.

Since every gridded permutation either avoids or contains the set H, the set $\mathcal{A}$ is the disjoint union of $\mathcal{B}$ and $\mathcal{C}$. Moreover, the formal definition of Requirement Insertion requires that both $\mathcal{B}$ and $\mathcal{C}$ are nonempty in order for it apply. Both parts of Condition 2 follow immediately from these two facts. □

The above proof makes clear that once any strategy is known to be a disjoint-union-type strategy, it is guaranteed to be productive.

6.5.2 Obstruction and Requirement Simplification

The equivalence strategies defined in Subsection 6.3.2 — Obstruction Deletion, Requirement Deletion, and Requirement List Deletion — were specifically defined to only apply to a tiling when the corresponding alteration did not change the underlying set of gridded permutations. We discussed in this section that as a result, these three strategies are in some sense trivial. However, they allow us to detect that the tiling representation of a set of gridded permutations can be simplified, which is advantageous for detecting equality of sets.

To prove that these three strategies are in fact equivalence strategies, one would need to show that whenever $d_S(\mathcal{A}) = \mathcal{B}$, we have $|\mathcal{A}_n| = |\mathcal{B}_n|$ for all n. Since these strategies do not even change the underlying set of gridded permutations, we have by definition the much stronger equality $\mathcal{A} = \mathcal{B}$.

6.5.3 Point Placement

The justification that the point placement strategy is an equivalence strategy is technical and requires quite a bit of bookkeeping. The material in this subsection is independent of the rest of this work, and so readers who wish to skip this subsection on their first reading may freely do so.

Consider the tiling $\mathcal{T} = ((t, u), \mathcal{O}, \mathcal{R})$ with $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$, and suppose $\mathcal{R}_1 = \{r\}$. We will describe the tiling $\mathcal{T}''$ that results from placing the point $r(I)$ of the requirement r in the extreme d direction for some $d \in \{\leftarrow, \rightarrow, \uparrow, \downarrow\}$.

Point placement results in the row and column of the placed point, $r(I)$, being split into three rows and three columns, with the cell containing the placed point itself becoming nine cells. For the rest of this section, suppose the placed point is in cell $c = (c_x, c_y)$. The effect of this splitting on the gridded permutations in $\mathcal{T}$ can be thought of pictorially in the following way: Take any gridded permutation $g \in \text{Grid}(\mathcal{T})$, and add two new vertical lines in column c_x and two new horizontal lines in row c_y such that none of these four lines intersect any of the entries of g. The result is a gridded permutation in $\mathcal{G}^{(t+2,u+2)}$. Figure 25 shows an example of this procedure. Given a gridded permutation g, we will now define the set of *multiplexes* of g around cell c,

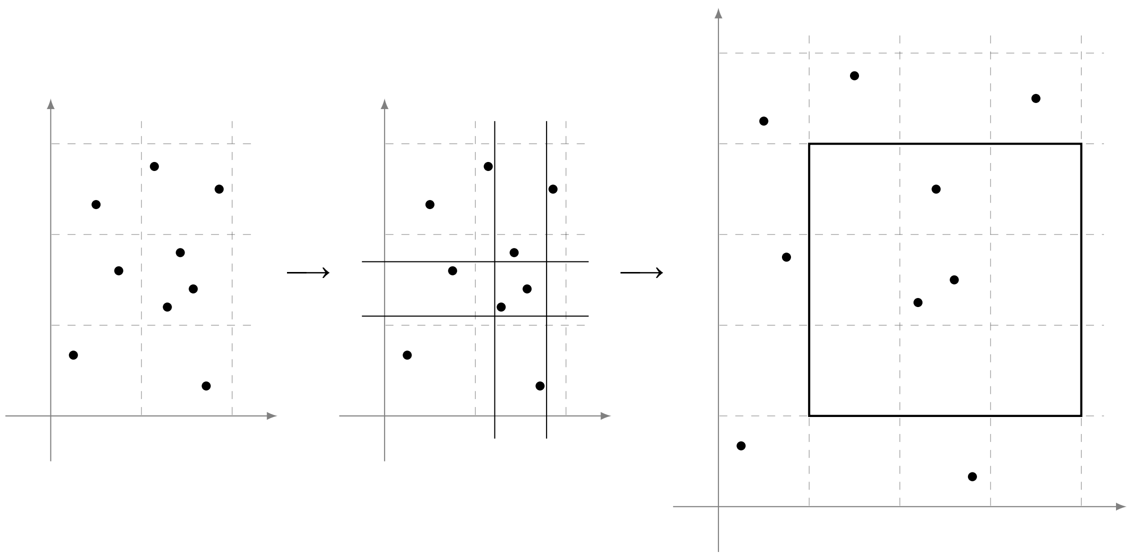


Figure 25: On the left, a gridded permutation g of size 9. In the center, the same gridded permutation with two vertical lines drawn in column $c_x = 1$ and two horizontal lines drawn in row $c_y = 1$. On the right, the resulting gridded permutation, in which the outlined 3×3 region corresponds to the single cell $(c_x, c_y) = (1, 1)$ which has now been split. This rightmost gridded permutation (which has the same underlying permutation, but different cell assignments) is called a *multiplex* of g.

denoted $M_c(g)$, to be the set of all gridded permutations in $\mathcal{G}^{(t+2,u+2)}$ formed by adding two vertical and two horizontal lines through c in this way.

Given a multiplexed gridded permutation $g' \in M_c(g)$, the original gridded permutation g can be recovered by, essentially, removing the two vertical lines between columns c_x, $c_x + 1$, and $c_x + 2$ and removing the two horizontal lines between rows c_y, $c_y + 1$, and $c_y + 2$. In this context we call g the *contraction* of g' around c. To make this more formal, consider the following map:

$$b_p : i \mapsto \begin{cases} i & i < p \\ p & i \in \{p, p+1, p+2\} \\ i-2 & i > p+2 \end{cases} .$$

With this terminology, we can define what contracting a gridded permutation around the cell $c = (c_x, c_y)$ does to the cell that each entry lies in:

$$\beta_{(c_x,c_y)} : (u, v) \mapsto (b_{c_x}(u), b_{c_y}(v)).$$

The reader is encouraged to refer back to Figure 25 which has $c = (1, 1)$, and in this case, for example, $\beta_{(1,1)}(0, 1) = (0, 1)$, $\beta_{(1,1)}(0, 2) = (0, 1)$, $\beta_{(1,1)}(1, 4) = (1, 2)$, and $\beta_{(1,1)}(3, 2) = (1, 1)$.

We can now formally define the multiplex of $g = (\pi, (c_1, \ldots, c_n))$ around the cell c to be those gridded permutations whose contractions are equal to g:

$$M_c(g) = \{g' = (\pi, (c'_1, \ldots, c'_n)) \in \mathcal{G} : \beta_c(c'_i) = c_i \text{ for all } i \text{ such that } 1 \leqslant i \leqslant n\}.$$

Extending this notation to sets of gridded permutations, we define

$$M_c(S) = \bigcup_{g \in S} M_c(g).$$

It is important to note that because $M_c(g)$ can be thought of as the preimage of g under the contraction map, it follows that if $g \neq h$, then $M_c(g) \cap M_c(h) = \varnothing$.

Before moving on, we will state and prove two pattern preservation lemmas.

Lemma 6.4. Fix two gridded permutations g and h and a cell c. If $h \leqslant g$, then every multiplex of g contains some multiplex of h, i.e., for all $g' \in M_c(g)$, there exists some $h' \in M_c(h)$ such that $h' \leqslant g'$.

Proof. Fix a cell c and gridded permutations $g = (\pi, (c_1, \ldots, c_n))$ and $h = (\sigma, (d_1, \ldots, d_k))$. Suppose that g contains h at the indices $i_1, \ldots, i_k$, implying that $\pi(i_1) \ldots \pi(i_k)$ is order-isomorphic to σ and $c_{i_\ell} = d_\ell$ for $1 \leqslant \ell \leqslant k$.

Let $g' \in M_c(g)$ be arbitrary. We can write $g' = (\pi, (c'_1, \ldots, c'_n))$, with the property that $\beta_c(c'_i) = c_i$ for all i. Define h' to be the gridded permutation that g' contains at the indices $i_1, \ldots, i_k$. The underlying permutation of h' is still σ, so $h' = (\sigma, (c'_{i_1}, \ldots, c'_{i_k}))$. Because we must have $\beta_c(c'_{i_\ell}) = c_{i_\ell} = d_\ell$, it follows that $h' \in M_c(h)$, completing the proof. □

Lemma 6.5. Fix two gridded permutations g and h and a cell c. If some multiplex of g contains some multiplex of h, then $h \leqslant g$.

Proof. Fix a cell c and gridded permutations g and h. Suppose that $g' \in M_c(g)$ and $h' \in M_c(h)$ are such that g' contains h' at the indices $i_1, \ldots, i_k$. Thus we can write $g' = (\pi, (c'_1, \ldots, c'_n))$ and $h' = (\sigma, (c'_{i_1}, \ldots, c'_{i_k}))$ where π contains σ at the same indices. This implies that $g = (\pi, (\beta_c(c'_1), \ldots, \beta_c(c'_n)))$ and $h = (\sigma, (\beta_c(c'_{i_1}), \ldots, \beta_c(c'_{i_k})))$, from which we see that g contains h at the indices $i_1, \ldots, i_k$. □

We are now ready to define an intermediate tiling $\mathcal{T}'$ derived from our original tiling $\mathcal{T} = ((t, u), \mathcal{O}, \{\{r\}, \mathcal{R}_2, \ldots, \mathcal{R}_k\})$. The tiling $\mathcal{T}'$ will contain the subset of multiplexes (around cell (c_x, c_y)) of the permutations in Grid$(\mathcal{T})$ that contain exactly one point in the cell $(c_x + 1, c_y + 1)$, and no other points in column $c_x + 1$ or row $c_y + 1$. To that end, define $\mathcal{O}' = M_c(\mathcal{O})$ and $\mathcal{R}'_i = M_c(\mathcal{R}_i)$ for $2 \leqslant i \leqslant k$. To ensure that cell $(c_x + 1, c_y + 1)$ contains exactly one point, we define additional obstructions $A = \{(12, (c_x + 1, c_y + 1)), (21, (c_x + 1, c_y + 1))\}$ and an additional requirement list $C = \{(1, (c_x + 1, c_y + 1))\}$. To ensure that no other cells in column $c_x + 1$ or row $c_y + 1$ have any entries, we define yet more obstructions

$$B_1 = \{(1, (c_x + 1, j)) : 0 \leqslant j < u, j \neq c_y + 1\}$$

and

$$B_2 = \{(1, (i, c_y + 1)) : 0 \leqslant i < t, i \neq c_x + 1\}.$$

The tiling $\mathcal{T}'$ can now be defined:

$$\mathcal{T}' = ((t + 2, u + 2), \mathcal{O}' \cup A \cup B_1 \cup B_2, \{\mathcal{R}'_2, \ldots, \mathcal{R}'_k, C\}).$$

Note that the requirement r actually being placed has been dropped.

The set $\mathrm{Grid}(\mathcal{T}')$ contains some, but not all, of the multiplexes of gridded permutations in $\mathrm{Grid}(\mathcal{T})$ in a way that we will make more precise. In particular, the multiplexes in $\mathrm{Grid}(\mathcal{T}')$ are exactly those in which one entry in cell c has been isolated into its own cell (c_x+1, c_y+1), with no other points in that cell's row or column. Informally, these are the multiplexes that result from picking one point in cell c and adding the two vertical lines just to the left and just to the right of this point and adding the two horizontal lines just below and just above this point. The lemma below makes this more precise.

Lemma 6.6. Let $g \in \mathrm{Grid}(\mathcal{T})$ and suppose that g has precisely ℓ points in cell $c = (c_x, c_y)$. Then,

$$\left|M_c(g) \cap \mathrm{Grid}(\mathcal{T}')\right| = \ell.$$

Proof. Suppose that $g \in \mathrm{Grid}(\mathcal{T})$ and consider $g' \in M_c(g)$. Because g avoids all obstructions in $\mathcal{O}$, Lemma 6.5 shows that g' avoids all obstructions in $\mathcal{O}'$. Similarly, by Lemma 6.4, g' contains at least one requirement in each list $\mathcal{R}'_2, \ldots, \mathcal{R}'_k$.

It remains to show that if g contains ℓ entries in cell c, then there are precisely ℓ elements of $M_c(g)$ that avoid the obstructions in $A \cup B_1 \cup B_2$ and contain the requirement C. These obstructions together with the requirement collectively enforce the property that every gridded permutation in $\mathrm{Grid}(\mathcal{T}')$ has exactly one point in the cell (c_x+1, c_y+1), and no other points in the same row or column as this cell. For each of the ℓ entries in cell c of g, there is one multiplex g' of g that has that entry alone in cell (c_x+1, c_y+1) with no other entries sharing a column or row with this cell, and this g' is in $\mathrm{Grid}(\mathcal{T}')$.

Clearly, for any two of the ℓ entries of g in cell c, the multiplexed versions that isolate these entries are different. Therefore, $M_c(g) \cap \mathrm{Grid}(\mathcal{T}')$ contains ℓ gridded permutations, each of which has one of the ℓ entries in cell c of g isolated in cell (c_x+1, c_y+1) of g', with no other entries sharing a row or column. □

In the proof above, we discussed multiplexes of g that have a single entry isolated in cell $(c_x + 1, c_y+1)$ with no other entries sharing a row or column. These are relevant to the rest of this subsection, so we will call them the *point multiplexes* of g, and if the point $g(i)$ is in cell c of g, then we will specifically call the multiplex that isolates that point $m_c^i(g)$. The lowercase "m" is meant to convey that $m_c^i(g)$ is one single gridded permutation, while $M_c(g)$ is a set of gridded permutations.

Our goal now is to construct a tiling $\mathcal{T}''$ by adding obstructions and one requirement to $\mathcal{T}'$ with the result that $|M_c(g) \cap \mathrm{Grid}(\mathcal{T}'')| = 1$ for any $g \in \mathrm{Grid}(\mathcal{T})$. Recall that we are placing point $r(I)$ of the original requirement r in the extreme d direction. For g with ℓ entries in cell c, the obstructions and requirement will invalidate $\ell - 1$ of the point multiplexes, leaving only the one in which the isolated point is the point that, out of all points playing the role of $r(I)$ in an occurrence of r, is the one in the most extreme d direction.

First, we must add a requirement so that not only does every permutation in $\mathrm{Grid}(\mathcal{T}'')$ contain some mutiplex of the requirement r that we're placing, but more specifically contains the single point multiplex r' in which the isolated point of any gridded permutation in $\mathrm{Grid}(\mathcal{T}'')$ plays the role of $r'(I)$ in an occurrence of r. (This is not yet enforcing that this point is in the most extreme d direction.) To achieve this, we add the requirement list $\{m_c^I(r)\}$ as a new requirement list in $\mathcal{T}''$.

Lastly, we need to craft new obstructions to add to $\mathcal{T}''$ to enforce the property that the isolated point in any $g' \in \text{Grid}(\mathcal{T}'')$ is the point in the most extreme d direction that plays the role of $r(I)$ in any occurrence of r. This is actually simple. Consider the set of all point multiplexes of r. For any r' in this set with the property that the cell c_I in which the entry $r'(I)$ occurs is further the d direction than the cell $(c_x + 1, c_y + 1)$ that contains the isolated point, we add r' as an obstruction to $\mathcal{T}''$.

At this point, $\mathcal{T}''$ contains all obstructions and requirements of $\mathcal{T}'$ together with the new obstructions and the one new requirement described above. The tiling $\mathcal{T}''$ is the output tiling of the point placement strategy, and so we must now prove that there is a size-preserving bijection between $\text{Grid}(\mathcal{T})$ and $\text{Grid}(\mathcal{T}'')$. We start with the following lemma.

Lemma 6.7. Let $g \in \text{Grid}(\mathcal{T})$. Then,

$$\left|M_c(g) \cap \text{Grid}(\mathcal{T}'')\right| = 1.$$

Proof. Let $g \in \text{Grid}(\mathcal{T})$ and recall that $M_c(g) \cap \text{Grid}(\mathcal{T}')$ was shown to contain one point multiplex of g for each of the ℓ points in cell c of g. Of all the points in cell c of g that play the role of $r(I)$ in an occurrence of r (there is at least one such point because $\{r\}$ is a requirement list of $\mathcal{T}$), suppose the one in the most extreme d direction is located at index i. Then, by design, the point multiplex $m_c^i(g)$ is contained in $\text{Grid}(\mathcal{T}'')$. The other $\ell - 1$ point multiplexes of g are each isolating different points, and each of these other points either does not play the role of $r(I)$ in an occurrence of r, or does, but not in the most extreme d direction, and therefore they are precluded from $\text{Grid}(\mathcal{T}'')$ by virtue of the new obstructions enforcing extremeness of direction or the new requirement forcing the isolated point to play the role of $r(I)$. □

The lemma above allows us to define a map $\Gamma : \text{Grid}(\mathcal{T}) \to \text{Grid}(\mathcal{T}'')$ in which $\Gamma(g)$ is defined to be the single element of $M_c(g)$ in $\text{Grid}(\mathcal{T}'')$. Note that the cell c, the requirement r, the index of the placed point I, and the direction d are all implicit parameters of Γ.

Theorem 6.8. The map Γ defined above is a size-preserving bijection.

Proof. Lemma 6.7 ensures that Γ is well-defined, and the fact that Γ is size-preserving follows from the fact that every element of $M_c(g)$ has the same size of g. Moreover, Γ is injective as a result of the observation that if $g \neq h$ then $M_c(g) \cap M_c(h) = \varnothing$.

To see that Γ is surjective, let $g' \in \text{Grid}(\mathcal{T}'')$ and let g be such that $g' \in M_c(g)$, i.e, g is the contraction of g'. We will check that $g \in \text{Grid}(\mathcal{T})$, from which it follows that $\Gamma(g) = g'$.

Let o be any obstruction of $\mathcal{T}$ and recall that every mutiplex in $M_c(o)$ is an obstruction of $\mathcal{T}''$. Therefore, if g were to contain o, Lemma 6.4 would imply that g' contains a multiplex of o, a contradiction. Therefore, g avoids all obstructions of $\mathcal{T}$.

One of the requirements of $\mathcal{T}''$ is a point multiplex of the requirement r being placed, therefore g' contains this point multiplex. By Lemma 6.5, it follows that g satisfies the requirement list $\{r\}$.

Lastly, consider any of the other requirement lists $\mathcal{R}_i$ of $\mathcal{T}$ for $2 \leqslant i \leqslant k$. Since g' contains some element of $M_c(\mathcal{R}_i)$, Lemma 6.5 once again implies that g contains some element of $\mathcal{R}_i$.

Therefore, $g \in \operatorname{Grid}(\mathcal{T})$, confirming that Γ is a size-preserving bijection, and therefore the point placement strategy is an equivalence strategy. □

6.5.4 Row Separation and Column Separation

In Subsection 6.3.4 we defined row separation and column separation as equivalence strategies that can be applied when the cells in a row or column can be split into two separate rows or columns. We will discuss only row separation here, but the results all follow *mutatis mutandis* for column separation.

For the remainder of this subsection, let $\mathcal{T} = ((t,u), \mathcal{O}, \{\mathcal{R}_1, \ldots, \mathcal{R}_k\})$ be a tiling. Suppose there is a row r with a subset S of nonempty cells such that row separation applies. From the definition of row separation in Subsection 6.3.4, this means that if we set S' to be the nonempty cells in row r that are not in S, then for every pair of cells $(c_1, c_2) \in S \times S'$, if c_1 is to the left of c_2 then $(21, (c_1, c_2)) \in \mathcal{O}$ and if c_1 is to the right of c_2 then $(12, (c_2, c_1)) \in \mathcal{O}$. We call these gridded permutations the *critical row patterns* and denote the set of them by C.

In order to formally define the tiling $\mathcal{T}'$ produced by the row separation strategy, we first need to define a mapping $\gamma_{r,S}$ that describes how row separation moves entries around between cells. To that end, define

$$\gamma_{r,S} : (i,j) \mapsto \begin{cases} (i,j), & j < r \text{ or } (j = r \text{ and } i \in S), \\ (i,j+1), & j > r \text{ or } (j = r \text{ and } i \notin S), \end{cases}.$$

Let $[N]$ denote the set $\{1, \ldots, N\}$. The domain of $\gamma_{r,S}$ is $[t] \times [u]$. The range of $\gamma_{r,S}$ contains all cells in $[t] \times [u+1]$ except several in rows r and $r+1$, and those cells that it doesn't contain will contain point obstructions in $\mathcal{T}'$ (i.e., they will be empty). We define the codomain of $\gamma_{r,S}$ to be its range.

Now, the tiling $\mathcal{T}'$ that results from applying Row Separation to row r and cells S is formed by applying the map $\gamma_{r,S}$ to the cells in each obstruction and requirement of $\mathcal{T}$. To that end, define the map $\Gamma_{r,S}$ on gridded permutations by

$$\Gamma_{r,S}((\pi, (c_1, \ldots, c_n))) = (\pi, (\gamma_{r,S}(c_1), \ldots, \gamma_{r,S}(c_n))).$$

From here on out we will refer to $\gamma_{r,S}$ and $\Gamma_{r,S}$ just as γ and Γ, as r and S are fixed throughout.

We must take a moment to verify that for any gridded permutation g, the definition of $\Gamma(g)$ actually produces a valid gridded permutation when g does not contain any of the critical row patterns.

Lemma 6.9. If g avoids all of the critical row patterns, then the definition of $\Gamma(g)$ above produces a valid gridded permutation.

Proof. The concern to address is whether the underlying pattern is consistent with the new cell assignments. First, note that the column assignments of every entry remain unchanged by γ. The row assignments stay largely unchanged except that some entries in row r move to row $r+1$.

This could pose a problem if g contained a 12 pattern in row r, but in $\Gamma(g)$ the 1 moves to row $r+1$ while the 2 stays in row r—clearly such a gridded permutation is impossible. A problem would similarly occur if g contained a 21 pattern in row r such that, again, the 1 moves to row $r+1$ under the effect of Γ while the 2 stays in row r. However, these two problematic patterns are precisely the critical row patterns, which g is assumed to avoid. ☐

We extend the definition of Γ to sets of gridded permutations under the definition $\Gamma(A) = \{\Gamma(g) : g \in A\}$.

We can finally give the formal definition of the tiling $\mathcal{T}'$ that results from row separation. We may assume without loss of generality that the obstructions $\mathcal{O}$ of $\mathcal{T}$ are pairwise incomparable. Now define

$$\mathcal{T}' = ((t, u+1), \Gamma(\mathcal{O} \setminus C) \cup \mathcal{O}', \{\Gamma(\mathcal{R}_1), \ldots, \Gamma(\mathcal{R}_k)\}),$$

where

$$\mathcal{O}' = \{(1, (i, r)) : i \notin S\} \cup \{(1, (i, r+1)) : i \in S\}$$

is a set of size 1 obstructions setting empty the cells in row r that correspond to columns not in S and the cells in row $r+1$ that correspond to columns that are in S.

To prove that Row Separation is an equivalence strategy, we must exhibit a size-preserving bijection between $\mathrm{Grid}(\mathcal{T})$ and $\mathrm{Grid}(\mathcal{T}')$. In fact, in addition to being helpful in the definition of $\mathcal{T}'$, Γ is such a bijection (with domain and codomain appropriately defined). The following lemma will help to establish this.

Lemma 6.10. The cell map γ is a bijection.

Proof. Consider two cells $c_1 = (i_1, j_1)$ and $c_2 = (i_2, j_2)$. Suppose that $\gamma(c_1) = \gamma(c_2)$ and let (a, b) denote this value. Since γ holds the first component constant, we must have $i_1 = a = i_2$. If $b \leqslant r$ then $j_1 = b = j_2$, and if $b \geqslant r+2$ then $j_1 = b-1 = j_2$. Lastly, if $b = r+1$, then j_1 and j_2 could equal r or $r+1$, but this depends only on the fixed set S and the quantities i_1 and i_2, which have already been shown to be equal. Therefore $j_1 = j_2$, and so γ is injective. The surjectivity of γ was ensured by our declaration that its codomain equals its range. ☐

We now state and prove two lemmas about the preservation of patterns under the Γ map that are similar to Lemmas 6.4 and 6.5 from Subsection 6.5.3.

Lemma 6.11. Consider a tiling $\mathcal{T}$ with row r and set of cells S in row r to which row separation applies. Let $h, g \in \mathrm{Grid}(\mathcal{T})$ with $h \leqslant g$. Then $\Gamma(h) \leqslant \Gamma(g)$.

Proof. Let h and g be as in the hypotheses and note that $\Gamma(h)$ and $\Gamma(g)$ are valid gridded permutations by Lemma 6.9. Suppose that g contains h at the indices $i_1, \ldots, i_k$, so we can write

$$g = (\pi, (c_1, \ldots, c_n)) \text{ and}$$
$$h = (\sigma, (c_{i_1}, \ldots, c_{i_k}))$$

where π contains σ at the same indices. Then,

$$\Gamma(g) = (\pi, (\gamma(c_1), \ldots, \gamma(c_n))) \text{ and}$$
$$\Gamma(h) = (\sigma, (\gamma(c_{i_1}), \ldots, \gamma(c_{i_k}))),$$

confirming that $\Gamma(g)$ contains $\Gamma(h)$ at the same indices. ☐

Lemma 6.12. Consider a tiling $\mathcal{T}$ with row r and set of cells S to which row separation applies. Let $h, g \in \operatorname{Grid}(\mathcal{T})$ and suppose $\Gamma(h) \leqslant \Gamma(g)$. Then, $h \leqslant g$.

Proof. Let h and g be as in the hypotheses and assume that $\Gamma(h) \leqslant \Gamma(g)$ at the indices $i_1, \ldots, i_k$. Writing

$$\begin{aligned} g &= (\pi, (c_1, \ldots, c_n)) \text{ and} \\ h &= (\sigma, (d_1, \ldots, d_k)), \end{aligned}$$

we have

$$\begin{aligned} \Gamma(g) &= (\pi, (\gamma(c_1), \ldots, \gamma(c_n))) \text{ and} \\ \Gamma(h) &= (\sigma, (\gamma(d_1), \ldots, \gamma(d_k))). \end{aligned}$$

As $\Gamma(g)$ contains $\Gamma(h)$ at the indices $i_1, \ldots, i_k$, we know that π contains σ at the same indices and that $\gamma(c_{i_\ell}) = \gamma(d_\ell)$ for $1 \leqslant \ell \leqslant k$. Since γ is a bijection, we have $c_{i_\ell} = d_\ell$ for each ℓ, and therefore g contains h also at the indices $i_1, \ldots, i_k$. □

With these lemmas in hand, we are ready to prove that row separation is an equivalence strategy.

Theorem 6.13. Row separation (and correspondingly, column separation) are equivalence strategies.

Proof. In order to establish the theorem, we will show that $\Gamma : \operatorname{Grid}(\mathcal{T}) \to \operatorname{Grid}(\mathcal{T}')$, with Γ as defined earlier, is a size-preserving bijection. Let us note right away that Γ does not change the underlying pattern of a gridded permutation, it simply affects the cells that the pattern lies in; this makes clear that Γ is size-preserving.

We first consider whether Γ is validly defined. Let $g \in \operatorname{Grid}(\mathcal{T})$ and define $g' = \Gamma(g)$. Since g avoids the critical row patterns Lemma 6.9 ensures that g' is a validly defined gridded permutation.

Next we will demonstrate that $g' \in \operatorname{Grid}(\mathcal{T}')$. The obstructions of $\mathcal{T}'$ are split into those of the form $\Gamma(\sigma)$ where σ is an obstruction of $\mathcal{T}$ that is not a critical row pattern, and the size 1 obstructions defined as $\mathcal{O}'$. The definition of Γ ensures that g' will not contain any points in the cells that $\mathcal{O}'$ requires be empty, and the contrapositive of Lemma 6.12 shows that for any non-critical row pattern σ, since g avoids σ we must have that g' avoids $\Gamma(\sigma)$. Similarly, Lemma 6.11 shows that for each requirement list $\mathcal{R}_i$, since g contains $\mathcal{R}_i$ we must have that g' contains $\Gamma(\mathcal{R}_i)$. Thus, $h \in \operatorname{Grid}(\mathcal{T}')$.

For injectivity, let $g_1 = (\pi_1, (c_1, \ldots, c_n))$ and $g_2 = (\pi_2, (d_1, \ldots, d_n))$. By the definition of Γ,

$$\begin{aligned} \Gamma(g_1) &= (\pi_1, (\gamma(c_1), \ldots, \gamma(c_n))), \text{ and} \\ \Gamma(g_2) &= (\pi_2, (\gamma(d_1), \ldots, \gamma(d_n))). \end{aligned}$$

Suppose that $\Gamma(g_1) = \Gamma(g_2)$. Then, since Γ does not affect the underlying pattern, $\pi_1 = \pi_2$, and we will use π to denote this permutation. Moreover, $\gamma(c_i) = \gamma(d_i)$ for all i, and so by the injectivity of γ it follows that $c_i = d_i$ for all i. Thus $g_1 = g_2$, verifying that Γ is injective.

To see that Γ is surjective, let $g' = (\pi, (d_1, \ldots, d_n)) \in \mathrm{Grid}(\mathcal{T}')$. Every cell d_i is in the range of γ because the cells in $[t] \times [u+1]$ that are not in the range of γ contain size 1 obstructions implying that no gridded permutation can have entries in those cells. Therefore, we may define $g = (\pi, (\gamma^{-1}(d_1), \ldots, \gamma^{-1}(d_n)))$ from which it is clear that $\Gamma(g) = g'$. □

6.5.5 Factor

In this subsection, we must demonstrate the productivity of the Factor strategy $\texttt{Factor}_{P,S}$. As discussed in an example on page 52, the reliance profile function for the Factor strategy was carefully defined in terms of the set S to ensure productivity. For ease of exposition, we prove productivity in the case of a partition P into two parts. Productivity for larger partitions follows inductively.

Theorem 6.14. The Factor strategy is productive.

Proof. Suppose $d_{\texttt{Factor}_{P,S}}(\mathcal{A}) = (\mathcal{B}^{(1)}, \mathcal{B}^{(2)})$. Recall that $S \subseteq \{1,2\}$, and if $i \in S$ then $\mathcal{A}_n$ does not rely on $\mathcal{B}^{(i)}_n$. We consider the cases $S = \varnothing$, $S = \{1\}$, and $S = \{1,2\}$, while the fourth case $S = \{2\}$ is symmetric to the case $S = \{1\}$.

To prove productivity, we have to check Conditions 1 and 2 in Definition 4.2 on page 31. Condition 1 requires that $\mathcal{A}_n$ can never rely on $\mathcal{B}^{(1)}$ or $\mathcal{B}^{(2)}$ at some size longer than n, and that is true in all cases by the definition of the reliance profile function for the Factor strategy. Condition 2 requires that for each $i \in \{1,2\}$, if $\mathcal{A}_N$ relies on $\mathcal{B}^{(i)}_N$ for some $N \in \mathbb{N}$, then:

(a) $|\mathcal{A}_n| \geqslant |\mathcal{B}^{(i)}_n|$ for all $n \in \mathbb{N}$, and

(b) $|\mathcal{A}_\ell| > |\mathcal{B}^{(i)}_\ell|$ for some $\ell \in \mathbb{N}$.

Case 1. Suppose $S = \{1,2\}$. In this case, $\mathcal{A}_n$ relies on neither $\mathcal{B}^{(1)}_n$ nor $\mathcal{B}^{(2)}_n$, and so Condition 2 trivially holds.

Case 2. Suppose $S = \{1\}$. This means that $|\mathcal{B}^{(2)}_0| = 0$ and that $\mathcal{A}_n$ does not rely on $\mathcal{B}^{(1)}_n$. We therefore only need to verify Condition 2 for $\mathcal{B}^{(2)}$. First note that for all $n \in \mathbb{N}$,

$$|\mathcal{A}_n| = |\mathcal{B}^{(1)}_0||\mathcal{B}^{(2)}_n| + |\mathcal{B}^{(1)}_1||\mathcal{B}^{(2)}_{n-1}| + \cdots + |\mathcal{B}^{(1)}_{n-1}||\mathcal{B}^{(2)}_1|$$

and that all terms on the right-hand side are nonnegative integers. In particular, $|\mathcal{B}^{(1)}_0| \geqslant 1$ because $2 \notin S$. Thus $|\mathcal{A}_n| \geqslant |\mathcal{B}^{(2)}_n|$ for all n, verifying Condition 2(a). To check Condition 2(b), let $m_1 > 0$ be minimal such that $|\mathcal{B}^{(1)}_{m_1}| > 0$ and let $m_2 > 0$ be minimal such that $|\mathcal{B}^{(2)}_{m_2}| > 0$; the existence of m_1 and m_2 are guaranteed by the requirement in the definition of the Factor strategy that $\mathcal{B}^{(1)}$ and $\mathcal{B}^{(2)}$ each contain at least one object of size at least 1. Now, we have

$$|\mathcal{A}_{m_1+m_2}| \geqslant |\mathcal{B}^{(1)}_0||\mathcal{B}^{(2)}_{m_1+m_2}| + |\mathcal{B}^{(1)}_{m_1}||\mathcal{B}^{(2)}_{m_2}|,$$

and because $|\mathcal{B}^{(1)}_0|$, $|\mathcal{B}^{(1)}_{m_1}|$ and $|\mathcal{B}^{(2)}_{m_2}|$ are all at least 1, we have

$$|\mathcal{A}_{m_1+m_2}| > |\mathcal{B}^{(2)}_{m_1+m_2}|,$$

verifying Condition 2(b).

Case 3. Suppose $S = \varnothing$. In this case, we are guaranteed that both $|\mathcal{B}_0^{(1)}| \geqslant 1$ and $|\mathcal{B}_0^{(2)}| \geqslant 1$. Condition 2 must now be checked for both $\mathcal{B}^{(1)}$ and $\mathcal{B}^{(2)}$. We omit the details, as they are substantially similar to Case 2. □

6.5.6 Obstruction Inferral

As was the case with Obstruction and Requirement Simplification in Subsection 6.5.2, there are no details that need to be checked. By definition, the Obstruction Inferral strategy only applies in cases where the underlying set of gridded permutations is not modified at all, guaranteeing by definition that it is an equivalence strategy.

6.6 Recap

This concludes our discussion applying Combinatorial Exploration to the domain of permutation patterns. In this work, we described only six (families of) strategies that together can successfully find combinatorial specifications for many permutation classes. However, we have developed a handful of additional, more complicated strategies that we will describe in future work, and which permit the discovery of combinatorial specifications for many more classes, including many listed in Subsection 2.4. To give just a taste here, among these additional strategies, a few examples are

- ◇ *row placement*, which is a disjoint-union-type strategy that splits a tiling into multiple children depending on whether a given row is empty, or if nonempty, which cell in the row has the smallest (or largest) entry; similarly, *column placement*;
- ◇ *fusion*, which detects when two rows or columns can be merged together into one row or column. This is informally like deleting the line that separates two rows, and this is the strategy that introduces the catalytic variables discussed in Subsection 2.4;
- ◇ several novel verification strategies that detect when the enumeration of a particular tiling can be computed independently, often by running Combinatorial Exploration again in a subprocess on this particular tiling.

7. Applying Combinatorial Exploration to Alternating Sign Matrices

7.1 Alternating Sign Matrices

An *alternating sign matrix* (ASM) is a matrix in which every entry is 0, 1, or -1, the sum of every row and every column is 1, and the nonzero entries in each row and in each column alternate between 1 and -1. These conditions imply that every alternating sign matrix is a square matrix.

Permutations can be thought of as matrices whose entries are either 0 or 1 and in which every row and every column contains a single 1, and thus ASMs are a generalization of permutations.

The enumeration of ASMs was first completed by Zeilberger [140] who proved that the number of $n \times n$ ASMs is

$$\prod_{k=0}^{n-1} \frac{(3k+1)!}{(n+k)!}.$$

Kuperberg [108] and Fischer [80] later gave shorter proofs.

The containment relation of permutations extends naturally to ASMs and pattern avoidance has been studied on these objects. Johansson and Linusson [96] defined what it means for an ASM to avoid a permutation as a pattern (which we will repeat below) and computed the enumeration of 132-avoiding ASMs. In this brief section, we will show how Combinatorial Exploration can be applied to ASMs and describe a handful of strategies that would be sufficient to recover the enumeration of the 132-avoiding ASMs.

Definition 7.1. To any permutation π of size n we associate the $n \times n$ matrix with a 1 in each entry $(i, \pi(i))$ and a 0 elsewhere.

We choose to index our matrices with Cartesian coordinates to match the standard way of depicting permutations. For example, the permutation matrix of 132 is $\begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$.

Definition 7.2. An ASM M is said to *contain* a permutation π if it is possible to form the permutation matrix of π from M by deleting rows, deleting columns, and changing some -1 and 1 entries into 0 entries. Otherwise M is said to *avoid* π.

This definition will be generalized below after further definitions have been made.

7.2 Gridded and Partial ASMs

In order to perform Combinatorial Exploration, we need to decide which combinatorial sets to work with and how they will be represented (the analogue of tilings), and then define strategies that decompose these sets. Like in the case of permutations, we will actually work with a gridded version of the objects at hand[17]. Because this section, and the several that follow, are meant to serve just as "proof-of-concepts" for Combinatorial Exploration, we will not attempt to make the definitions of the objects and strategies involved completely formal.

Definition 7.3. A *gridded ASM* is an ASM in which the columns and rows have been partitioned into contiguous, possibly-empty parts. Less formally, a gridded ASM can be formed by taking an ASM and adding any number of vertical lines either between columns or to the extreme left or extreme right, then doing the same with horizontal lines to rows. Like with gridded permutations, we will refer to the rectangular regions as *cells*.

For example, $\left[\begin{array}{c|c|cc|c|c} & & 0 & 0 & 1 & 0 \\ \hline & & 0 & 1 & 0 & 0 \\ & & 1 & 0 & -1 & 1 \\ & & 0 & 0 & 1 & 0 \\ \hline & & & & & \end{array}\right]$ is a gridded ASM whose underlying ASM is $\begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$.

Cell $(0, 2)$ of the gridded ASM contains no entries, and cell $(2, 1)$ contains six entries.

Because we will be applying strategies that pull ASMs apart into smaller matrices (similar to the "factor" strategy for permutations), we also need to define a generalization of an ASM that does not enforce the row and column sum conditions.

Definition 7.4. A *partial ASM* is a (not necessarily square) matrix whose entries are all 0, 1,

[17]This is not a requirement to use Combinatorial Exploration, but we have found it effective. Our penchant for geometrizing objects may be a result of our own extensive experience working with gridded permutations.

and -1 with the property that the nonzero entries in each row and column alternate between -1 and 1. It is not required that the entries in each row and column sum to 1, and thus it is permitted for the leftmost or rightmost nonzero entry in a row to be -1, and similarly for columns. *Gridded partial ASMs* are defined in the analogous way.

We can now extend Definition 7.2 by defining when one partial ASM contains another.

Definition 7.5. A partial ASM M_1 is said to *contain* a partial ASM M_2 if it is possible to form M_2 from M_1 by deleting rows, deleting columns, and changing some -1 and 1 entries into 0 entries. Otherwise, M_1 is said to *avoid* M_2.

This containment relation extends in the natural way to containment of gridded partial ASMs just as in the case of gridded permutations.

7.3 ASM-Tilings

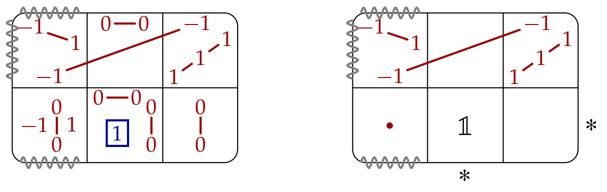
Having now defined the combinatorial objects involved, we will define an analogue of tilings that represent sets of gridded (partial) ASMs and can be easily manipulated with combinatorial strategies. We will keep our definition rather informal, prioritizing understanding over formality because this section is only meant to serve as a proof-of-concept.

Definition 7.6. An *ASM-tiling* $\mathcal{T}$ is a structure that represents a set of gridded partial ASMs. It is defined by four components. The first three are identical to the three components that define a gridded permutation tiling: dimensions (t, u), a set $\mathcal{O}$ of gridded partial ASMs called *obstructions*, and a set $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$ of sets of gridded partial ASMs called *requirements*. The gridded partial ASMs that $\mathcal{T}$ represents, $\text{Grid}(\mathcal{T})$, must each avoid all of the gridded partial ASMs in $\mathcal{O}$ and must each contain at least one gridded partial ASM in each $\mathcal{R}_i$. However, the fourth component in the definition adds a further condition that must be met by each element of $\text{Grid}(\mathcal{T})$.

The fourth component is a marking of each portion of the border of the ASM-tiling as either "partial" or "complete". For example, if the part of the border along the top of an ASM-tiling above column c is marked "partial" (pictorially, we will draw that part with an oscillating line), this represents that gridded partial ASMs M in $\text{Grid}(\mathcal{T})$ may, in any columns of M that correspond to column c in $\mathcal{T}$, have a topmost nonzero entry equal to -1 (which is permitted in partial ASMs, but not standard (complete) ASMs). Similarly, if the part of the border along the bottom of column c is marked "partial", then columns of M that correspond to column c in $\mathcal{T}$ may have a bottommost nonzero entry -1. Similar conditions apply to rows.

Consider the example ASM-tiling $\mathcal{T}$ shown on the left in Figure 26. The dimensions of $\mathcal{T}$ are $(t, u) = (3, 2)$, and there is a single requirement r that is the gridded ASM whose matrix consists of a single entry with the value 1 in cell $(1, 0)$. There are five obstructions that consist of two 0 entries. Avoiding two 0 entries vertically in a cell forces any gridded partial ASM drawn on $\mathcal{T}$ to have at most a single row of entries in that cell (and thus in the entire row of that cell in $\mathcal{T}$). Similarly, avoiding two 0 entries horizontally forces at most a single column of entries. As a result, the requirement r together with the five obstructions consisting of two zeros essentially isolate the "1" entry much like point placement does on gridded permutation tilings.

There are two size 1 obstructions in the bottom-left cell, which forbid any gridded partial ASM drawn on $\mathcal{T}$ from containing a non-zero entry gridded in that cell. The other three obstructions



$$\left[\begin{array}{ccc|c|cccc} 1 & 0 & -1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}\right]$$

Figure 26: An ASM-tiling $\mathcal{T}$ shown with all obstructions and requirements (left) and the same tiling (middle) shown with the convention that actual rows and columns have an asterisk ($*$) next to them, cells that can only contain 0 have a point obstruction, and placed entries are bold. On the right is a partial gridded ASM in Grid($\mathcal{T}$).

are shown with an abbreviated notation in which the 0s have been omitted. For example, the size 2 obstruction in the top-left corner forces any gridded partial ASM drawn on $\mathcal{T}$ to avoid the pattern $\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$ in its top-left cell. The size 3 obstruction in the top-right corner forces any gridded partial ASM draw on $\mathcal{T}$ to avoid the pattern $\begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$ in its top-right corner. Finally, the size 2 obstruction that crosses between two cells forces any gridded partial ASM drawn on $\mathcal{T}$ to not contain a $\begin{bmatrix} 0 & -1 \\ -1 & 0 \end{bmatrix}$ pattern in which the first column is in the top-left cell and the second column is in the top-right cell.

Lastly, in this example, three segments of the boundary of $\mathcal{T}$ have been designated "partial". This implies that any columns of a gridded partial ASM drawn on $\mathcal{T}$ that are gridded into the leftmost column of $\mathcal{T}$ are permitted to have a topmost entry equal to -1 and a bottommost entry equal to -1. Similarly, any rows that are gridded into the topmost row of $\mathcal{T}$ are permitted to have a leftmost entry equal to -1, but not a rightmost entry.

In the middle of Figure 26 the same tiling is shown with the convention that rows of $\mathcal{T}$ that can only contain a single row of any gridded partial ASM have an asterisk ($*$) on the boundary, and similarly for columns. Cells which can only contain 0 are marked with a point obstruction and placed entries are shown bold. On the right is one example of a gridded partial ASM that belongs to Grid($\mathcal{T}$).

7.4 ASMs Avoiding the Pattern 132

One result of Johansson and Linusson [96] is the enumeration of ASMs that avoid the permutation 132 (in the sense of Definition 7.2). Here we show that their result can be automatically discovered using Combinatorial Exploration on the objects described above. We see this as justification that a fully implemented version of Combinatorial Exploration in the ASM domain would be likely to discover many new results, including the open questions raised in [96].

We are also using this opportunity to preview a significant extension of Combinatorial Exploration that will be detailed in later work. Figures 27 and 28 depict a set of combinatorial rules that provide sufficient structural information to compute the enumeration of the 132-avoiding ASMs, but the reader will immediately notice the figures show, what appear to be, two separate proof trees, not one.

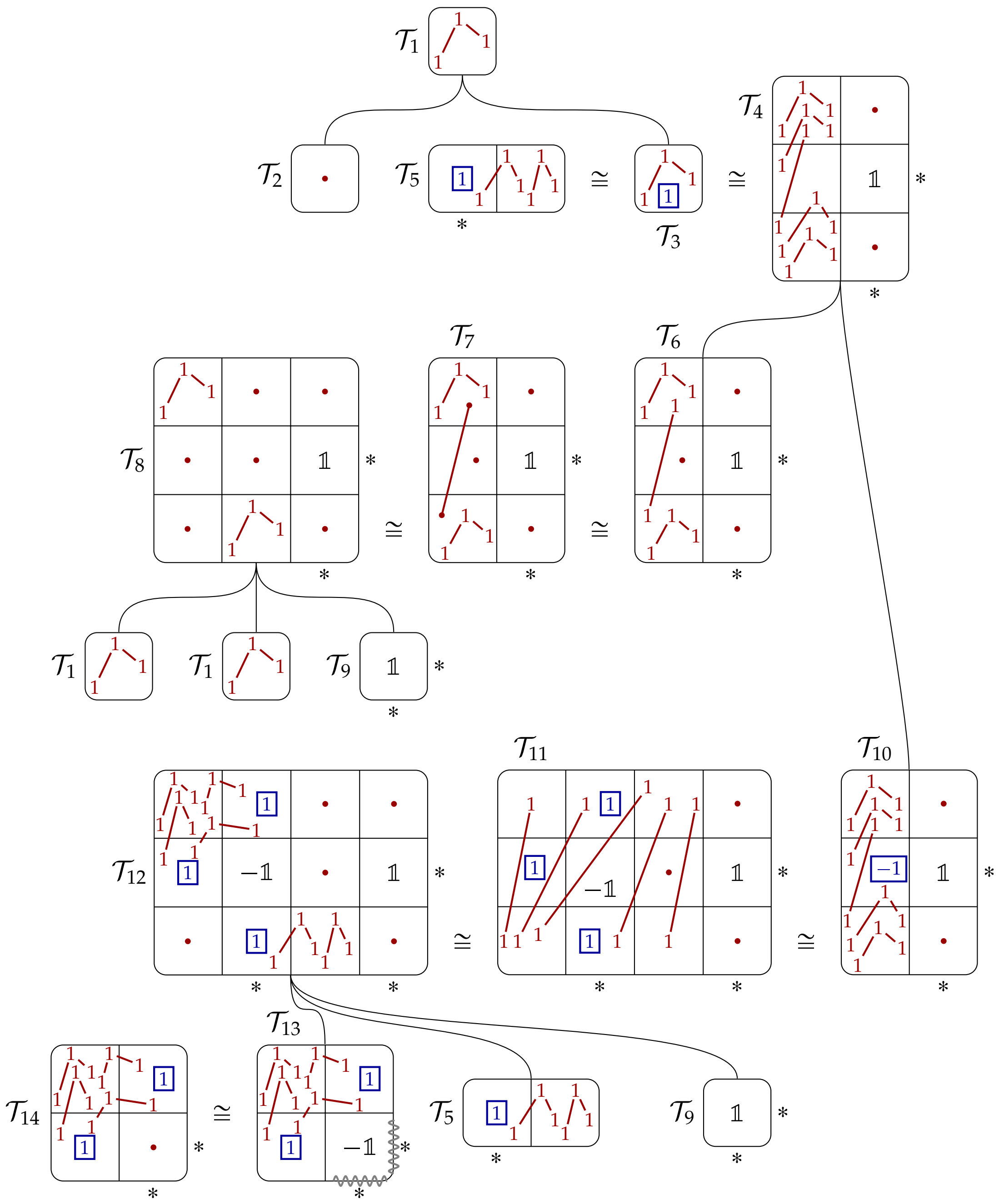


Figure 27: The first partial proof tree for 132-avoiding ASMs. Note that obstructions which use points should be interpreted to forbid both 1 and -1 entries, as in tiling $\mathcal{T}_7$. In some tilings, like $\mathcal{T}_{11}$ we have not shown all obstructions, only the most important ones.

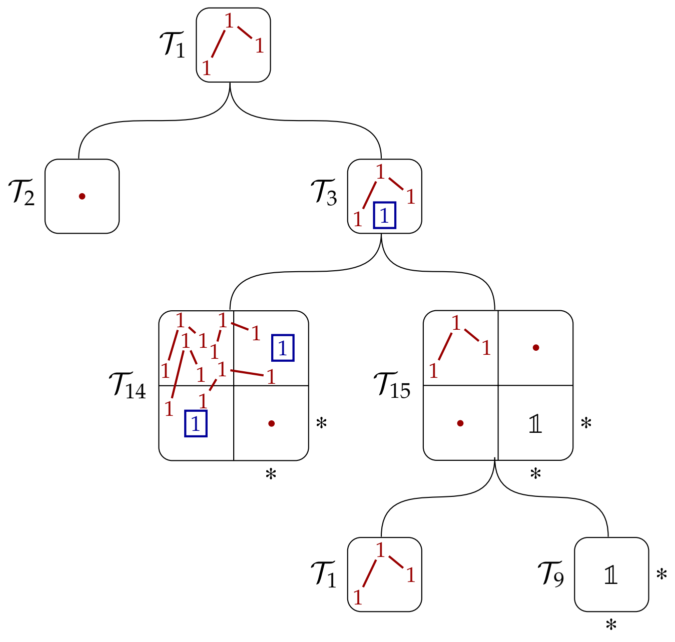


Figure 28: The second partial proof tree for 132-avoiding ASMs.

Neither of these are actually proof trees, because their corresponding combinatorial specifications do not have the property that every set appearing on a right-hand side also appears on exactly one left-hand side (in terms of the proof tree perspective, this condition requires that every set that is the child of a rule is also the parent of exactly one rule). So, while the set of combinatorial rules formed from taking both trees together is not a combinatorial specification, it still contains sufficient information to find the enumeration of all sets involved.[18] We call such a set of rules a *combinatorial forest*. These generalize combinatorial specifications and we will describe in future work how the Combinatorial Exploration algorithm can be modified to search for combinatorial forests, along with a productivity-style guarantee that they suffice to enumerate each of the sets involved.

We have surpressed some obstructions in the tilings in the two trees that are not relevant, like in tiling $\mathcal{T}_{11}$ in Figure 27. We use a point in an obstruction to mean either a 1 or a -1.

We will end this section by roughly describing the strategies that produce the rules shown in the two partial proof trees in Figures 27 and 28 and then solving the corresponding system of equations.

- ◇ There is a disjoint-union-type strategy similar to the requirement insertion strategy on gridded permutations that decomposes a set into two subsets depending on whether a specific gridded partial ASM is contained or avoided. The rule $\mathcal{T}_1 \leftarrow (\mathcal{T}_2, \mathcal{T}_3)$ inserts the gridded ASM $\left[\begin{array}{c} 1 \end{array}\right]$ as a requirement into cell $(0,0)$. The rule $\mathcal{T}_4 \leftarrow (\mathcal{T}_6, \mathcal{T}_{10})$ inserts the requirement $\left[\begin{array}{c|c} & \\ \hline -1 & \\ \hline & \end{array}\right]$.

[18]However, we would not be surprised if an implemented version of Combinatorial Exploration found a genuine combinatorial specification.

- ⋄ There are equivalence strategies $\mathcal{T}_3 \leftarrow (\mathcal{T}_4)$ and $\mathcal{T}_{10} \leftarrow (\mathcal{T}_{11})$ that are essentially the same as the point placement strategy for gridded permutations. The rule $\mathcal{T}_3 \leftarrow (\mathcal{T}_5)$ is also produced by a point-placement-style strategy, but in this case it does not isolate the 1 in its own row, just in its own column.

- ⋄ The rule $\mathcal{T}_3 \leftarrow (\mathcal{T}_{14}, \mathcal{T}_{15})$ is a disjoint-union-type strategy that places the "1" requirement depending on whether the rightmost 1 is in the bottom-right corner or not.

- ⋄ The rules $\mathcal{T}_2 \leftarrow ()$ and $\mathcal{T}_9 \leftarrow ()$ are verification strategies, as Grid$(\mathcal{T}_2)$ contains only the size 0 gridded ASM and Grid$(\mathcal{T}_9)$ contains only the size 1 ASM with the entry 1.

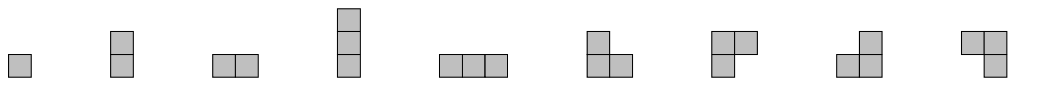
- ⋄ The equivalence rule $\mathcal{T}_6 \leftarrow (\mathcal{T}_7)$ recognizes that the presence of the obstruction $\left[\begin{array}{cc|c} 0 & 1 & \\ \hline & & \\ \hline 1 & 0 & \end{array}\right]$ crossing from the bottom-left cell to the top-left cell implies the existence of the three additional obstructions $\left[\begin{array}{cc|c} 0 & -1 & \\ \hline & & \\ \hline 1 & 0 & \end{array}\right]$, $\left[\begin{array}{cc|c} 0 & 1 & \\ \hline & & \\ \hline -1 & 0 & \end{array}\right]$, and $\left[\begin{array}{cc|c} 0 & -1 & \\ \hline & & \\ \hline -1 & 0 & \end{array}\right]$ that cross in the same way. This is because, for example, if the pattern $\left[\begin{array}{cc|c} 0 & 1 & \\ \hline & & \\ \hline -1 & 0 & \end{array}\right]$ occurred, the ASM row sum property would require the presence of a 1 to the left of the -1, which would then create the forbidden obstruction together with the 1 in the top-left cell. Similar logic leads to the rule $\mathcal{T}_{11} \leftarrow (\mathcal{T}_{12})$: if either of the cells containing a 1 obstruction contained a -1, then the ASM row and column sum properties would force them to also contain a 1, which is forbidden.

- ⋄ The equivalence rule $\mathcal{T}_7 \leftarrow (\mathcal{T}_8)$ is essentially the same as column separation for gridded permutations.

- ⋄ The equivalence rule $\mathcal{T}_{13} \leftarrow (\mathcal{T}_{14})$ observes that the mapping of taking a gridded partial ASM in Grid$(\mathcal{T}_{13})$ and changing the -1 in the bottom-right corner into a 0 is a size-preserving bijection to Grid$(\mathcal{T}_{14})$. Note that this conversion now ensures that all ASMs in Grid$(\mathcal{T}_{14})$ obey the row and column sum properties, while this was not true for those partial ASMs in Grid$(\mathcal{T}_{13})$.

- ⋄ Finally, there are three rules, $\mathcal{T}_8 \leftarrow (\mathcal{T}_1, \mathcal{T}_1, \mathcal{T}_9)$, $\mathcal{T}_{12} \leftarrow (\mathcal{T}_{13}, \mathcal{T}_5, \mathcal{T}_9)$, and $\mathcal{T}_{15} \leftarrow (\mathcal{T}_1, \mathcal{T}_9)$, that come from a strategy similar to the factor strategy of gridded permutations. The first and third rule are rather straight-forward: each cell of the parent is in its own row and column, and no parts of the border are marked as partial, and therefore, the cells of the parent can be split into the children shown, all of which also have no parts of their own borders marked as partial. The second rule is more intricate. Any place where one child shares a row or column with another child, those shared parts of the boundary must get marked as partial in the children. This explains the partial boundary of $\mathcal{T}_{13}$. At first it seems like $\mathcal{T}_5$ and $\mathcal{T}_9$ should also each have a part of their boundaries marked as partial (the left half of the top border of $\mathcal{T}_5$ and the left border of $\mathcal{T}_9$), but in each case, the -1 cell that stayed with $\mathcal{T}_{13}$ implies that the row and column sum conditions actually remain satisfied on $\mathcal{T}_5$ and $\mathcal{T}_9$.

From this combinatorial forest (the union of the rules in each of the partial proof trees), we



$$\begin{bmatrix}1\end{bmatrix} \begin{bmatrix}1\\1\end{bmatrix} \begin{bmatrix}1 & 1\end{bmatrix} \begin{bmatrix}1\\1\\1\end{bmatrix} \begin{bmatrix}1 & 1 & 1\end{bmatrix} \begin{bmatrix}1 & 0\\1 & 1\end{bmatrix} \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix} \begin{bmatrix}0 & 1\\1 & 1\end{bmatrix} \begin{bmatrix}1 & 1\\0 & 1\end{bmatrix}$$

Figure 29: Nonempty polyominoes with at most three squares and their matrix representations

obtain an algebraic system of equations as outlined in Section 5, which can be solved to discover that the generating function for $\mathcal{T}_1$ is

$$T_1(x) = \frac{3 - x - \sqrt{1 - 6x + x^2}}{2}$$

and therefore the counting sequence for the 132-avoiding ASMs is the large Schröder numbers:

$$1, 1, 2, 6, 22, 90, 394, 1806, 8558, \ldots.$$

8. Applying Combinatorial Exploration to Polyominoes

8.1 Polyominoes

In this section a *square* refers to a region $[i, i+1] \times [j, j+1]$ in the Euclidean plane. A *polyomino* is a finite union of squares with a connected interior. So, for example, two squares that only share a corner would not be a polyomino. Two polyominoes are considered equal if one can be transformed into the other by a vertical or a horizontal translation of the plane.[19] The word “cell” is more common for what we call a “square”, but we will use “cell” for another concept below.

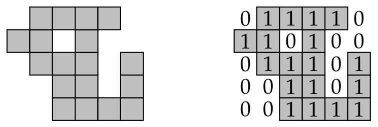
Polyominoes have been used for modeling physical phenomena such as crystal-growth (see Dhar [70]) and percolation (see Broadbent and Hammersley [47], Conway and Guttmann [65], and Rensburg [92]). Figure 29 shows the nonempty polyominoes with at most three squares.

There are at least two natural definitions of the size of a polyomino, each of which turns the set of polyominoes into a combinatorial set. These are *area*, the number of squares in the polyomino, as well as *perimeter*, the distance along the border. These two definitions are sometimes used simultaneously, leading to bivariate generating functions. It is worth noting that the total number of polyominoes of a given size (either area or perimeter) is unknown, a stark contrast to the case of permutations.[20]

However, as is the case with permutations and many other combinatorial objects, several subsets of polyominoes have been intensely studied, including those that are convex, directed and convex, column-convex, directed column-convex (see Bousquet-Mélou and Fédou [43] and

[19]Some texts consider two polyominoes equal if one can be obtained from the other by rotation, but we consider these to be different.

[20]If we take the area as the size function then the enumeration of polyominoes is $1, 1, 2, 6, 19, 63, 216, \ldots$ and is sequence A001168 on the OEIS. While the general formula for these numbers, $a(n)$, is unknown, it is known that the limit $a(n)^{1/n}$ exists (see Klarner [99]), and is estimated to be close to 4.06 (Jensen [94]).



$$\begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 30: A polyomino, with its matrix drawn on top and shown separately.

Bousquet-Mélou [41]), inscribed in a rectangle (see Goupil, Cloutier, and Nouboud [86]), tree-like (see Aleksandrowicz, Asinowski, Barequet [11]), centered convex, Z-convex, 4-stack, bi-centered (see Fedou, Rinaldi, and Drosini [77]), and L-convex (see Castiglione, Frosini, Restivo, and Rinaldi [61] and Guttmann and Kotesovec [88]).

Many of the subsets that have been studied can be defined as the set of polyominoes that avoid certain patterns, and in this section we present a proof-of-concept to demonstrate that Combinatorial Exploration can be effective in the domain of polyominoes as well. Later in this section, we will derive a proof tree for the set of Ferrers diagrams (whose enumeration is equal to the number of integer partitions) and describe, but not show, a proof tree for the L-convex polyominoes (also sometimes called moon polyominoes).

We start by using the matrix-based description of polyominoes given by Frosini, Guerrini, and Rinaldi [82] in which squares of a polyomino are represented by a 1 and the remaining locations of the bounding box are represented by a 0. Figure 29 shows the matrix representations for each polyomino with at most 3 squares, and Figure 30 shows a larger example. From this description, we can define when one polyomino contains another.

Definition 8.1 ([82])**.** A polyomino P_1 is said to *contain* a polyomino P_2 if it is possible to form the matrix for P_2 from the matrix for P_1 by deleting rows and deleting columns.

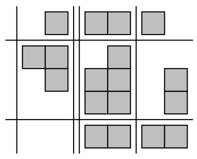
We may extend this definition slightly by allowing P_2 to be a matrix that does not satisfy the bounding box criteria, e.g., the polyominoes that avoid $P_2 = \begin{bmatrix} 1 & 0 \end{bmatrix}$ are those that do not have a square to the left of a non-square (within the bounding box of the polyomino). In this context, we call matrices such as P_2 *polyomino patterns*.

Following the development of Combinatorial Exploration in the domain of alternating sign matrices in Section 7, we will now define a gridded version of polyominoes and a tiling-like structure that represents the sets of gridded polyominoes upon which we will perform Combinatorial Exploration.

8.2 Gridded Polyominoes

Definition 8.2. A *gridded polyomino* is a polyomino in which the columns and rows have been partitioned into contiguous, possibly-empty parts. Less formally, a gridded polyomino can be formed by taking the matrix representation of a polyomino and adding any number of vertical lines either between columns or to the extreme left or extreme right, then doing the same with horizontal lines to rows. Like with gridded permutations and gridded ASMs, we will refer to the rectangular regions as *cells*. Figure 31 shows an example of a gridded polyomino.

The containment relation of polyominoes and the notion of a polyomino pattern extend in a



$$\left[\begin{array}{cc||cc|cc}
0 & 1 & 1 & 1 & 1 & 0 \\ \hline
1 & 1 & 0 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 1 & 0 & 1 \\ \hline
0 & 0 & 1 & 1 & 1 & 1
\end{array}\right]$$

Figure 31: A gridding of the polyomino from Figure 30.

natural way to gridded analogues in the same way as with gridded permutations.

8.3 Polyomino Tilings

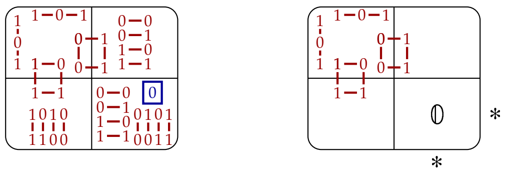
For the remainder of this section, we will focus on enumerating polyominoes by area (the number of squares); only small changes would be needed to set up Combinatorial Exploration to enumerate by perimeter.

Definition 8.3. A *poly-tiling* $\mathcal{T}$ is a structure that represents a set of gridded polyominoes. It is defined by four components. The first three are identical to the three components that define a gridded permutation tiling: dimensions (t, u), a set $\mathcal{O}$ of gridded polyomino patterns called *obstructions*, and a set $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$ of sets of gridded polyomino patterns called *requirements*. The gridded polyominoes that $\mathcal{T}$ represents, $\mathrm{Grid}(\mathcal{T})$, must each avoid all of the gridded polyomino patterns in $\mathcal{O}$ and must each contain at least one gridded polyomino pattern in each $\mathcal{R}_i$.

The fourth component, new to this domain, is called a *tracking list*. The tracking list designates a subset of the rows of the tiling. This component does not change which gridded polyominoes are in $\mathrm{Grid}(\mathcal{T})$. Instead, it signals to the combinatorial strategies that we are not just interested in the sequence a_n for the number of polyominoes in $\mathrm{Grid}(\mathcal{T})$ with area n, but the more refined sequence $a_{n,k}$ for the number of polyominoes in $\mathrm{Grid}(\mathcal{T})$ with area n and with k squares in the rows indicated by the tracking list. Informally, when applying strategies to a tiling, the strategy will look at the tracking list to determine whether they can be applied, and if so what the children will be. Correspondingly, the counting functions for the strategy will involve two indices, n and k.

This is a simplified version of the strategies briefly discussed in Subsection 2.4 that lead to catalytic variables in the systems of equations for generating functions, and which will be discussed further in future work.

Consider the example poly-tiling $\mathcal{T}$ shown on the left in Figure 32. The dimensions of $\mathcal{T}$ are $(t, u) = (2, 2)$, and there is a single requirement, the gridded polyomino pattern that contains 0 in the cell $(1, 0)$. The cells $(1, 0)$, $(1, 1)$ each contain the four obstructions $\begin{bmatrix}0 & 0\end{bmatrix}$, $\begin{bmatrix}0 & 1\end{bmatrix}$, $\begin{bmatrix}1 & 0\end{bmatrix}$, and $\begin{bmatrix}1 & 1\end{bmatrix}$. These force any gridded polyomino in $\mathrm{Grid}(\mathcal{T})$ to have at most one column gridded into those cells. The vertical versions of these four obstructions appear in cells $(0, 0)$, $(1, 0)$, and they enforce a similar condition on the rows. The eight obstructions and one requirement in cell $(1, 0)$ enforce that any gridded polyomino in $\mathrm{Grid}(\mathcal{T})$ has a 0 gridded into this cell and no other entries. There are four remaining obstructions, two local to the cell $(0, 1)$ and two crossing between cells. This tiling has an empty tracking list. The second subfigure in Figure 32 shows the same tiling with the same kind of abbreviations as we used for ASM-tilings. The proof tree for Ferrers diagrams in Figure 34 has many examples of tilings that do have



$$\left[\begin{array}{ccc|c} 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ \hline 1 & 0 & 1 & 0 \end{array}\right]$$

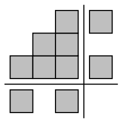


Figure 32: A tiling $\mathcal{T}$ with all obstructions and requirements shown explicitly; the same tiling with abbreviations; a gridded polyomino on the tiling in matrix form; the same polyomino in traditional form.

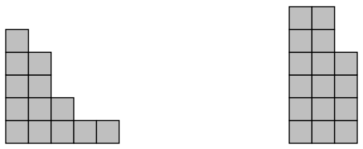


Figure 33: Two Ferrers diagrams.

tracking lists, shown by drawing an oscillating line around the rows in the list.

The third subfigure of Figure 32 shows the matrix representation of a gridded polyomino in $\mathrm{Grid}(\mathcal{T})$. It has three columns that have been gridded into the first column of $\mathcal{T}$ and one gridded into the second, and it has one row that has been gridded into the first row of $\mathcal{T}$, and three gridded into the second. The last subfigure of Figure 32 shows the same gridded polyomino in the traditional form.

8.4 Ferrers Diagrams

A Ferrers diagram is a polyomino in which all rows are aligned to the left side of the bounding box and each row is at least as big as the one above it. Figure 33 shows two examples of Ferrers diagrams. Ferrers diagrams with n squares are in bijection with integer partitions of n: the partition $p_1 \geqslant p_2 \geqslant \ldots \geqslant p_\ell$ of n maps to the Ferrers diagram whose row lengths, starting from the bottom, are $p_1, p_2, \ldots, p_\ell$. As a result, the enumeration of Ferrers diagrams with n squares is known, and the generating function satisfies an algebraic differential equation, but does not satisfy a linear differential equation.

Ferrers diagrams are precisely those polyominoes that avoid the polyomino patterns $\beta_1 = \begin{bmatrix} 0 & 1 \end{bmatrix}$ and $\beta_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$. Figure 34 shows a proof tree for the set of Ferrers diagrams. As with other domains, we have used visual abbreviations to simplify the pictures of poly-tilings.

We will now briefly explain the strategies that produce each rule of the tree.

- ⋄ The rule $\mathcal{T}_1 \leftarrow (\mathcal{T}_2, \mathcal{T}_3)$ is produced by a requirement-insertion-style strategy that splits into the case where the polyomino is empty and the case where it contains at least one square.
- ⋄ A point-placement-style strategy places the requirement in $\mathcal{T}_3$ first as bottommost as possible, then as rightmost as possible, to get $\mathcal{T}_3'$ (some implicit simplifications have been applied, just as we often did in the permutation domain). A different strategy observes that the rightmost column in $\mathcal{T}_3'$ cannot contain any entries, as it would then contain a β_2

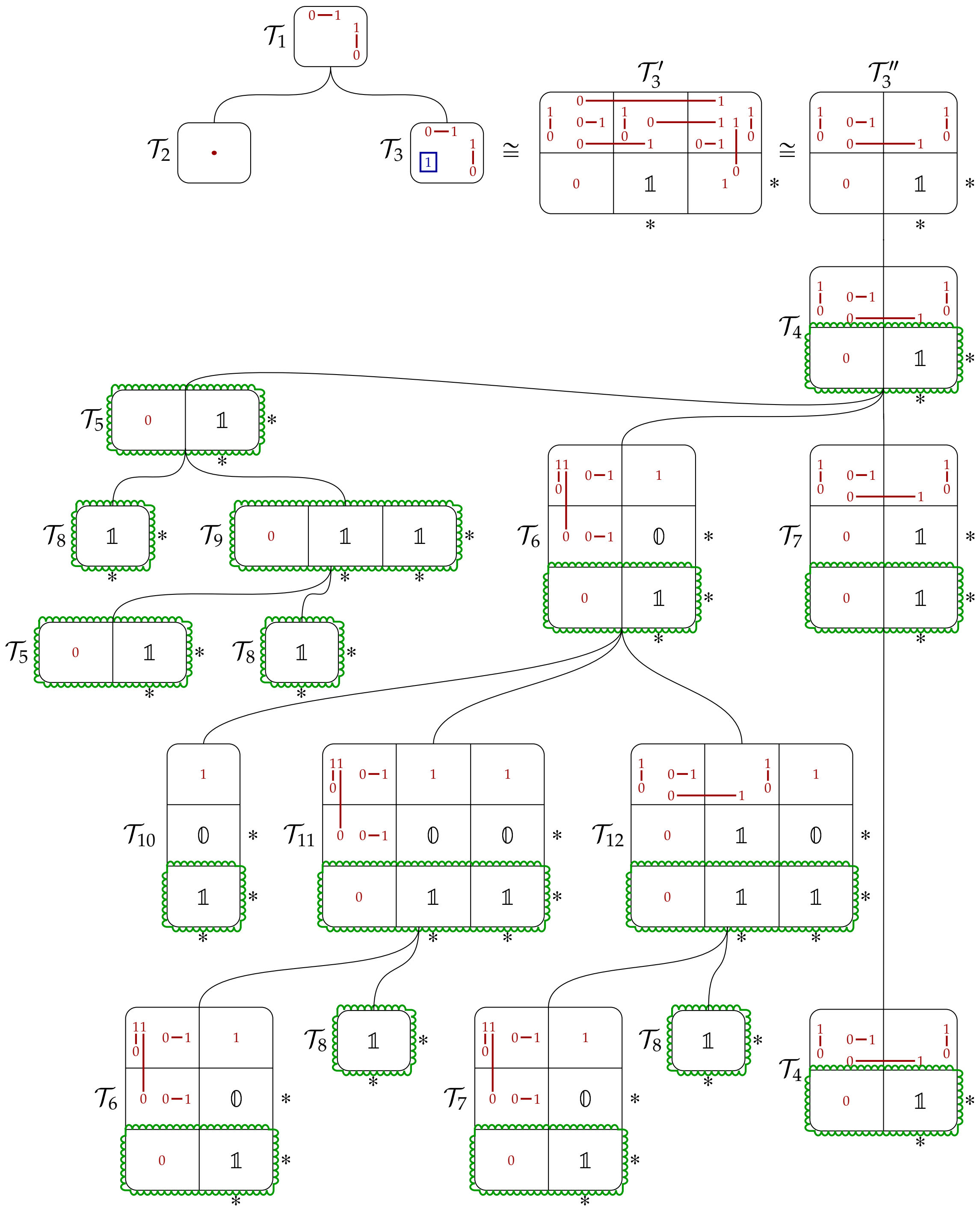


Figure 34: A proof tree for Ferrers diagrams.

pattern, and therefore $\mathcal{T}_3$ and $\mathcal{T}_3''$ are equinumerous.

- ◇ The rule $\mathcal{T}_3'' \leftarrow (\mathcal{T}_4)$ simply adds the tracking list for the bottommost row. Although it may at first feel backward, this is a valid strategy because we can certainly obtain the counting sequence for $\mathcal{T}_3''$ if we know the counting sequence for $\mathcal{T}_4$; indeed, we just ignore the extra index corresponding to the tracking list.
- ◇ The rules $\mathcal{T}_4 \leftarrow (\mathcal{T}_5, \mathcal{T}_6, \mathcal{T}_7)$, $\mathcal{T}_5 \leftarrow (\mathcal{T}_8, \mathcal{T}_9)$, and $\mathcal{T}_6 \leftarrow (\mathcal{T}_{10}, \mathcal{T}_{11}, \mathcal{T}_{12})$ are disjoint-union-type strategies that split into cases depending on what the extreme value in a particular cell in a row or column of height or width 1 is. For example, in the first rule mentioned above, either the top row of $\mathcal{T}_4$ contains no entries at all (giving $\mathcal{T}_5$) or the entry of that row that is as far down and to the right as possible is a 0 (giving $\mathcal{T}_6$), or it is a 1 (giving $\mathcal{T}_7$).
- ◇ The rules $\mathcal{T}_9 \leftarrow (\mathcal{T}_5, \mathcal{T}_8)$, $\mathcal{T}_{11} \leftarrow (\mathcal{T}_6, \mathcal{T}_8)$, and $\mathcal{T}_{12} \leftarrow (\mathcal{T}_7, \mathcal{T}_8)$ are similar to the factor strategy on permutations. They detect when a particular square placed into a row and column of height and width 1 can be removed from every gridded polyomino on the tiling without violating the connectedness property.
- ◇ Two verification strategies give the rules $\mathcal{T}_2 \leftarrow ()$ and $\mathcal{T}_8 \leftarrow ()$ because their generating functions are easily seen to be 1 and xy, respectively. A more complicated verification strategy gives the rule $\mathcal{T}_{10} \leftarrow ()$ by detecting that there are no gridded polyominoes in $\text{Grid}(\mathcal{T}_{10})$ because the placed 0 will lead to a violation of the bounding box condition, i.e., that the topmost and bottommost rows and the leftmost and rightmost columns all contain at least one square.
- ◇ Lastly, and most interestingly, the rule $\mathcal{T}_7 \leftarrow (\mathcal{T}_4)$ is produced by a strategy that notices that the two bottommost rows of $\mathcal{T}_7$ must be identical. Therefore, they can be "merged" together. To see that this is a valid strategy, let $a_{n,k}$ denote the number of gridded polyominoes in $\text{Grid}(\mathcal{T}_7)$ with n squares, k of which are in the tracked row, and let $b_{n,k}$ be the corresponding sequence for $\text{Grid}(\mathcal{T}_4)$. Then, we have the counting formula $a_{n,k} = b_{n-k,k}$ and the generating function equation $A(x,y) = B(x,xy)$.

As discussed in Section 5, this proof tree provides a polynomial-time counting algorithm for the number of Ferrers diagrams of size n. Moreover, the system of equations for the generating functions of the combinatorial sets is the following.[21]

$$\begin{aligned}
T_1(x) &= T_2(x) + T_3(x) & T_7(x,y) &= T_4(x,xy)\\
T_2(x) &= 1 & T_8(x,y) &= xy\\
T_3(x) &= T_4(x,1) & T_9(x,y) &= T_5(x,y)\cdot T_8(x,y)\\
T_4(x,y) &= T_5(x,y) + T_6(x,y) + T_7(x,y) & T_{10}(x,y) &= 0\\
T_5(x,y) &= T_8(x,y) + T_9(x,y) & T_{11}(x,y) &= T_6(x,y)\cdot T_8(x,y)\\
T_6(x,y) &= T_{10}(x,y) + T_{11}(x,y) + T_{12}(x,y) & T_{12}(x,y) &= T_7(x,y)\cdot T_8(x,y)
\end{aligned}$$

This system can be solved to find the known generating function

$$T_1(x) = \prod_{i=1}^{\infty} \frac{1}{1-x^i}.$$

[21]The variable y tracks the number of squares of each gridded polyomino that are in rows identified by the tracking list of the tiling.

We have also derived, by hand, a proof tree for the large class of *L-convex polyominoes*, which also sometimes go by the name *moon polyominoes*. These can be defined by avoiding the four polyomino patterns

$$\begin{bmatrix}1 & 0 & 1\end{bmatrix}, \qquad \begin{bmatrix}1\\0\\1\end{bmatrix}, \qquad \begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}, \qquad \begin{bmatrix}0 & 1\\1 & 0\end{bmatrix},$$

and their enumeration is also already known as sequence A126764 in the OEIS [128]. Like with Ferrers diagrams, their generating function is non-D-finite. The proof tree is too large to show here, involving 47 combinatorial sets. Nevertheless, it serves as further proof that Combinatorial Exploration would be effective in the domain of polyominoes.

9. Applying Combinatorial Exploration to Pattern-Avoiding Set Partitions

9.1 Set Partitions

A set partition of size n is a decomposition of the set $[n] = \{1, \ldots, n\}$ into a set of nonempty disjoint subsets $B_1, \ldots, B_k$ called *blocks* whose union is $[n]$. For readability, we write set partitions without commas and braces, using / to separate the blocks. The order of the blocks and the order within the blocks do not matter, so for example the set partitions 134/28/5/79 and 97/82/341/5 are considered the same; the first of these is written in *standard form*, in which the entries within the blocks are written in increasing order, and the blocks are written in increasing order of their smallest elements. The five set partitions of size 3, written in standard form, are

$$123, \qquad 1/23, \qquad 12/3, \qquad 13/2, \qquad 1/2/3.$$

An alternative way to express a set partition $\sigma = B_1/\cdots/B_k$ is with a word $w = w(1)\ldots w(n)$ over the positive integers such that $w(i) = j$ if $i \in B_j$. The five partitions of size 3 can be expressed in this way as

$$111, \qquad 122, \qquad 112, \qquad 121, \qquad 123.$$

Words of this kind are called *restricted growth functions* and can be defined by the conditions

1. $w(1) = 1$,
2. for all $i \geqslant 2$, $w(i) \leqslant 1 + \max(\{w(1), \ldots, w(i-1)\})$.

There have been several different notions of pattern avoidance studied for set partitions. For example, the definition used by Klazar [100–102] and more recently Bloom and Saracino [34], Chen, Deng, Du, Stanley, and Yan [62], Riordan [125], and Touchard [132] says that σ contains τ if τ can be obtained by deleting entries from blocks of σ and standardizing the remaining entries. With this notion, the set partition 145/23 contains 12/34 as can be seen by deleting the 1 and standardizing (and rearranging the blocks, which is permitted because the blocks of a set partition are unordered). A second definition involving arc-diagrams has been studied as well by, for example Bloom and Elizalde [33].

In this section we work with a third notion of pattern avoidance that uses the correspondence with restricted growth functions. This notion was investigated by Sagan [127] (who also considered Klazar's version of pattern avoidance) and then studied further by Campbell, Dahlberg, Dorward, Gerhard, Grubb, Purcell, and Sagan [60], Dahlberg, Dorward, Gerhard, Grubb, Purcell, Reppuhn, and Sagan [68], and Jelínek, Mansour, and Shattuck [93]. In this third version of pattern avoidance, we say that σ contains τ if there is a (not necessarily consecutive) subsequence of $w(\sigma)$ that is order-isomorphic to $w(\tau)$. In this notion, it is no longer true that 145/23 contains 12/34 because the corresponding restricted growth functions are 12211 and 1122, and the former does not contain the latter as a subsequence. As in previous sections, we can generalize this containment notion to words that are not themselves restricted growth functions, e.g., the set partition 12211 contains the word 21. As before, we will call these words that are not necessarily actual set partitions *patterns*.

At the end of this section, we will give two proof trees, one for set partitions avoiding the pattern 1212 and the other for set partitions avoiding the pattern 111. The set partitions avoiding 1212 are often called *non-crossing* set partitions, and we will re-discover that they are counted by the Catalan numbers. Set partitions avoiding 111 have each part of size at most 2; for these we write down a system of differential equations and solve to re-discover that the generating function is D-finite.

9.2 Gridded Set Partitions

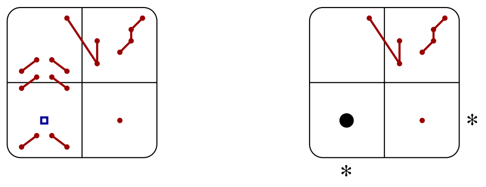
As in the previous sections, we will define a gridded version of set partitions, an analogous version of pattern containment, and then define a tiling-like object to represent sets of gridded set partitions. The easiest way to do this for this short proof-of-concept is to think of a set partition as a 0/1 matrix that contains a 1 in the (i, j) entry (indexed with Cartesian coordinates) if block B_i, written in standard form, contains the entry j. All other entries are 0. For example, the set partition 134/2 is represented by the matrix $\begin{bmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$.

All such matrices have exactly one 1 in each row. The restricted growth function property can be phrased in the following way: the $(1, 1)$ entry must be 1, and for all $i > 1$, if the lowest 1 in column i is at location (i, j), then subcolumn consisting of the bottommost $j - 1$ entries of column $i - 1$ must contain at least one 1.

Having defined the matrix representation of a set partition, we can now lean on previous sections by defining a gridded set partition as simply the matrix representation with vertical and horizontal lines added in the usual way. Gridded set partition patterns (those that do not necessarily obey the restricted growth function condition) are defined similarly, and the definition of pattern avoidance can be easily extended to these gridded analogues.

9.3 Set Partition Tilings

Definition 9.1. An *SP-tiling* $\mathcal{T}$ is a structure that represents a set of gridded set partitions. It is defined by three components, identical to those that define a gridded permutation tiling: dimensions (t, u), a set $\mathcal{O}$ of gridded set partition patterns called *obstructions*, and a set $\mathcal{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}$ of sets of gridded set partition patterns called *requirements*. The gridded set parti-



$$\left[\begin{array}{c|cccc} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 \end{array}\right]$$

Figure 35: An SP-tiling with all obstructions and requirements shown, followed by an abbreviated versions, and finally a gridded set partition on the tiling with underlying set partition 13/2/4/57/6. The corresponding restricted growth function is 1213454.

tions that $\mathcal{T}$ represents, $\text{Grid}(\mathcal{T})$, must each avoid all of the gridded set partition patterns in $\mathcal{O}$ and must each contain at least one gridded set partition pattern in each $\mathcal{R}_i$.

Consider the example SP-tiling $\mathcal{T}$ shown on the left in Figure 35. The dimensions of $\mathcal{T}$ are $(t, u) = (2, 2)$, and there is a single requirement, the size 1 gridded set partition in the cell $(1, 0)$. Like gridded permutation tilings, we can depict obstructions and requirements using dots with lines between them to signify the locations of the 1s. For example, the obstruction in the top-right corner is the gridded set partition 1223 (in restricted growth function notation) with all entries in the cell $(1, 1)$.

The six size 2 obstructions, together with the size 1 requirement, force every gridded set partition in $\text{Grid}(\mathcal{T})$ to have at most one column gridded into the first column of $\mathcal{T}$. The size 1 obstruction forces all gridded set partitions in $\text{Grid}(\mathcal{T})$ to have no entries gridded into cell $(1, 0)$ (although this is actually already implied by the constraint that the matrix of a set partition has exactly one 1 per row). Lastly, there is a crossing size 3 obstruction with underlying word 221 and the previously mentioned size 4 obstruction with underlying word 1223.

The middle of Figure 35 shows the visual abbreviations that we use with SP-tilings: we draw an asterisk below a column to show that it has width 1, similarly for rows, and use a solid black point to show a cell with precisely one element of the set partition. The right side of the figure shows a gridded set partition in $\text{Grid}(\mathcal{T})$.

9.4 Non-Crossing Set Partitions

Figure 36 shows a proof tree for the set partitions that avoid the pattern 1212. The strategies used are simple and will be familiar after having read the section on strategies for gridded permutations.

- ⋄ The rules $\mathcal{T}_1 \leftarrow (\mathcal{T}_2, \mathcal{T}_3)$ and $\mathcal{T}_3' \leftarrow (\mathcal{T}_4, \mathcal{T}_6)$ are produced by requirement insertion strategies. The first inserts into cell $(0, 0)$ and the second inserts into cell $(0, 1)$.
- ⋄ The rules $\mathcal{T}_4 \leftarrow (\mathcal{T}_1, \mathcal{T}_5)$ and $\mathcal{T}_6''' \leftarrow (\mathcal{T}_1, \mathcal{T}_3', \mathcal{T}_5)$ are produced by a factor-like strategy that splits apart cells that do not interact in their rows and columns and have no obstructions or requirements crossing between them.
- ⋄ The rules $\mathcal{T}_3 \leftarrow (\mathcal{T}_3')$ and $\mathcal{T}_6 \leftarrow (\mathcal{T}_6')$ place a requirement into its own row, much like the point placement strategy for permutations.

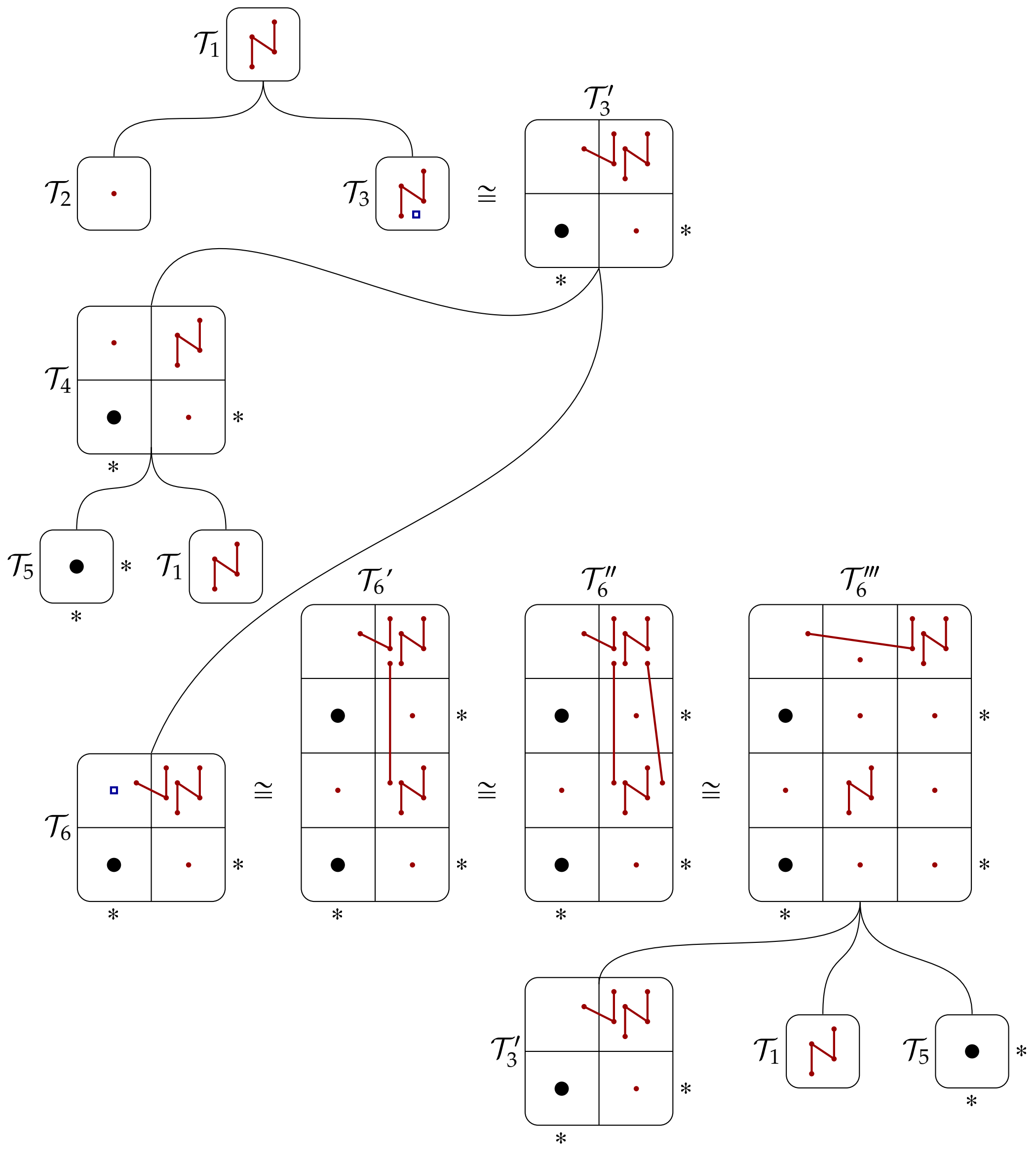


Figure 36: A proof tree for non-crossing set partitions.

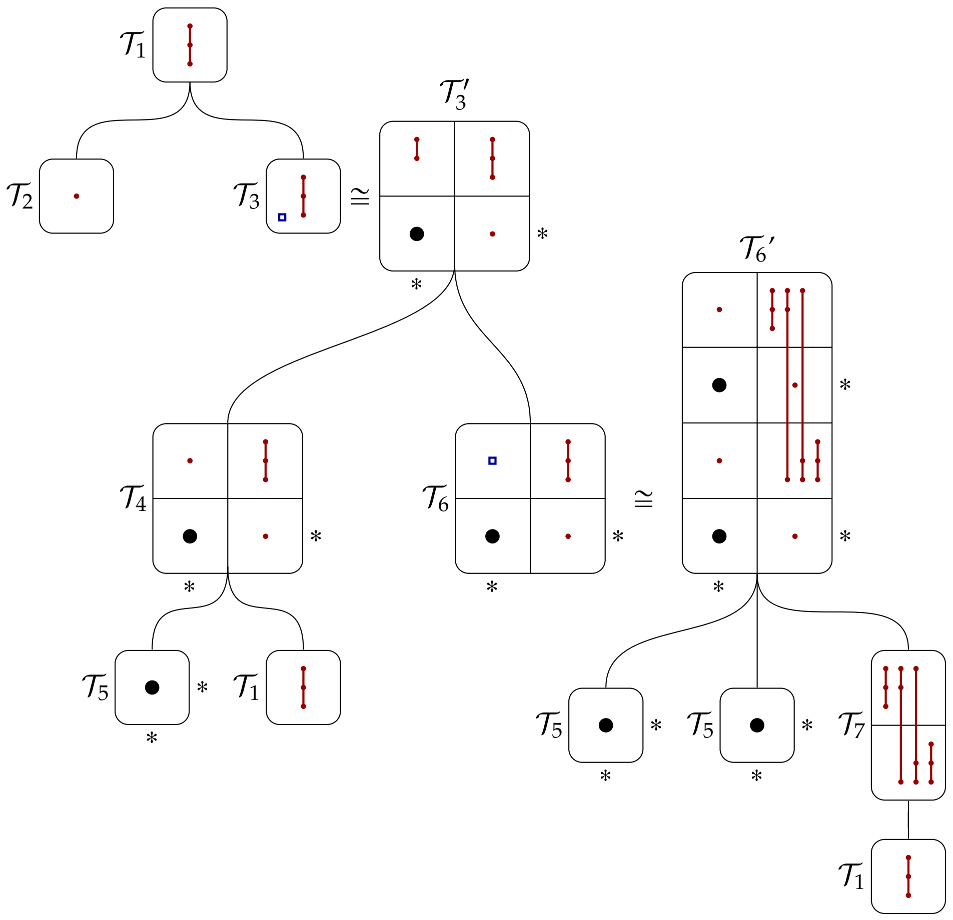


Figure 37: A proof tree for set partitions with parts of size at most 2.

- ⋄ The rule $\mathcal{T}_6' \leftarrow (\mathcal{T}_6'')$ infers that all gridded set partitions in Grid$(\mathcal{T}_6')$ also avoid the 21 obstruction crossing between the cells $(1,3)$ and $(1,1)$, as a result of the vertical size 2 obstruction and the restricted growth function property. Because of this new obstruction the rule $\mathcal{T}_6'' \leftarrow (\mathcal{T}_6''')$ is produced by a column separation strategy.

From this proof tree, we can write down an algebraic system of equations whose solution identifies that $T_1(x)$ is the Catalan generating function as expected.

9.5 Set Partitions with Parts of Size at Most 2

Figure 37 shows a proof tree for the set partitions that avoid the pattern 111. They are the set partitions with the property that every block has size 1 or 2. The ordinary generating function for their counting sequence is D-finite, but not algebraic, and we will see how this proof tree allows the defining linear differential equation to be computed. This sequence is A000085 in the OEIS [128].

All of the rules in this proof tree except for $\mathcal{T}_7 \leftarrow (\mathcal{T}_1)$ are produced by requirement insertion, point placement, and factor strategies as seen in the previous example. The rule $\mathcal{T}_7 \leftarrow (\mathcal{T}_1)$ is produced by a strategy similar to the "fusion" strategy mentioned at the end of Section 6.

The gridded set partitions in $\mathrm{Grid}(\mathcal{T}_7)$ can be formed uniquely by starting with a gridded set partition in $\mathrm{Grid}(\mathcal{T}_1)$ and adding a horizontal line between any pair of entries, or at the extreme bottom and top. In this way, a gridded set partition of size n in $\mathrm{Grid}(\mathcal{T}_1)$ produces $n+1$ gridded set partitions of size n in $\mathrm{Grid}(\mathcal{T}_7)$. This leads to the generating function equation $T_7(x) = \dfrac{d}{dx}(xT_1(x))$.[22]

The entire system of equations is

$$\begin{aligned}
T_1(x) &= T_2(x) + T_3(x)\\
T_2(x) &= 1\\
T_3(x) &= T_4(x) + T_6(x)\\
T_4(x) &= T_1(x) \cdot T_5(x)\\
T_5(x) &= x\\
T_6(x) &= T_5(x)^2 \cdot T_7(x)\\
T_7(x) &= \frac{d}{dx}\left(x \cdot T_1(x)\right).
\end{aligned}$$

We can deduce from this that

$$T_1(x) = 1 + (x + x^2)T_1(x) + x^3 \frac{d}{dx} T_1(x),$$

proving that $T_1(x)$ is D-finite.

10. Further Algorithmic Considerations

In this section we aim to give a small taste of the extensive algorithmic design work required to actually implement Combinatorial Exploration in an effective manner. Our implementation of Combinatorial Exploration, which we call the `CombSpecSearcher`, is open-source and can be found on Github [22]. Our code is fully compatible with the alternative Python implementation called `PyPy`, a free just-in-time compiler that in our experience tends to speed up computations by a factor of around 5 at the cost of some increased memory usage. The `CombSpecSearcher` is completely domain-agnostic, making no reference to permutations or any other combinatorial objects. In order to employ the `CombSpecSearcher` to perform Combinatorial Exploration, one must implement their own Python classes and functions representing their combinatorial objects and strategies, and plug these into the `CombSpecSearcher`. For more details about how this is done, one can refer to the "README" file in the `CombSpecSearcher` repository [22] which shows a sample implementation of Combinatorial Exploration for the simple domain of binary words. Additionally, our `Tilings` repository [21] on Github contains our implementation of Combinatorial Exploration for permutations discussed in Section 6.

The user initiates Combinatorial Exploration by specifying the combinatorial set that they wish to be enumerated, and a set of strategies to be used, which we call a *strategy pack*. The strategy pack separates the strategies into several different groups depending on how one wishes them to be used and in what order. The groups are called *inferral strategies*, *initial strategies*, *expansion strategies*, and *verification strategies*.

[22] Alternatively, one can write an algebraic system of equations that uses two catalytic variables, but no derivatives.

The `CombSpecSearcher` was designed with the *separation of concerns* design principle in mind, making the software as flexible as possible for future development. Work is delegated to a handful of separate components that work together to manage the flow of Combinatorial Exploration. The most important of these components, which we discuss in more detail below, are

- the `CSSQueue`, which manages the order in which combinatorial sets are decomposed by strategies;
- the `ClassDB`, which is in charge of storing the combinatorial sets discovered, including their compression and decompression to improve memory usage;
- the `EquivDB`, a union-find data structure that tracks the equivalence classes of combinatorial sets;
- the `RuleDB`, which stores all combinatorial rules discovered during the exploration process;
- the `SpecFinder`, which searches for a subset of combinatorial rules that form a combinatorial specification.

The `CSSQueue` controls the order in which combinatorial sets are considered and in which the strategies in the strategy pack are applied to them. It fundamentally has two operations: a method for adding a combinatorial set to the queue, and a method that tells `CombSpecSearcher` which combinatorial set should be expanded next and, moreover, which strategy should be used for this expansion. The default `CSSQueue` is actually composed of three separate queues: the *working queue*, the *current queue*, and the *next queue*.

When a combinatorial set is added to the `CSSQueue` for the first time, it is added to the working queue. When `CombSpecSearcher` is ready for a new combinatorial set and requests one from the `CSSQueue`, the `CSSQueue` first takes from the working queue if it is nonempty. Each combinatorial set $\mathcal{A}$ taken from the working queue is immediately expanded using the strategies in the pack that are designated as inferral strategies. These are equivalence strategies for which, if they apply, then we no longer want to try applying any additional strategies to the parent $\mathcal{A}$. For permutations, the row and column separation strategies discussed in Subsection 6.3.4 is effective as an inferral strategy—if a tiling has two columns that separate, we want that separation to occur right away, and then there is no need to do further work on the unseparated version. If an inferral strategy applies, the child set is added to the working queue and $\mathcal{A}$ is removed from the `CSSQueue` permanently.

If no inferral strategy successfully applies, $\mathcal{A}$ is then expanded using the strategies in the pack that are designated as initial strategies. The children of any strategy that applies are placed in the working queue, and after $\mathcal{A}$ has been expanded with all initial strategies, it is moved to the next queue. Good candidates for initial strategies are those that decompose a set into simpler sets that are likely to already exist in the universe like, for instance, the factor strategy of Subsection 6.3.5. Treating the factor strategy as an initial strategy means we will attempt to factor a tiling long before we apply other strategies that tend to push deeper into the universe, like the requirement insertion strategy of Subsection 6.3.1.

If the working queue is empty, the next combinatorial set to be expanded will be taken from the current queue. When a combinatorial set is taken from the current queue, it is expanded

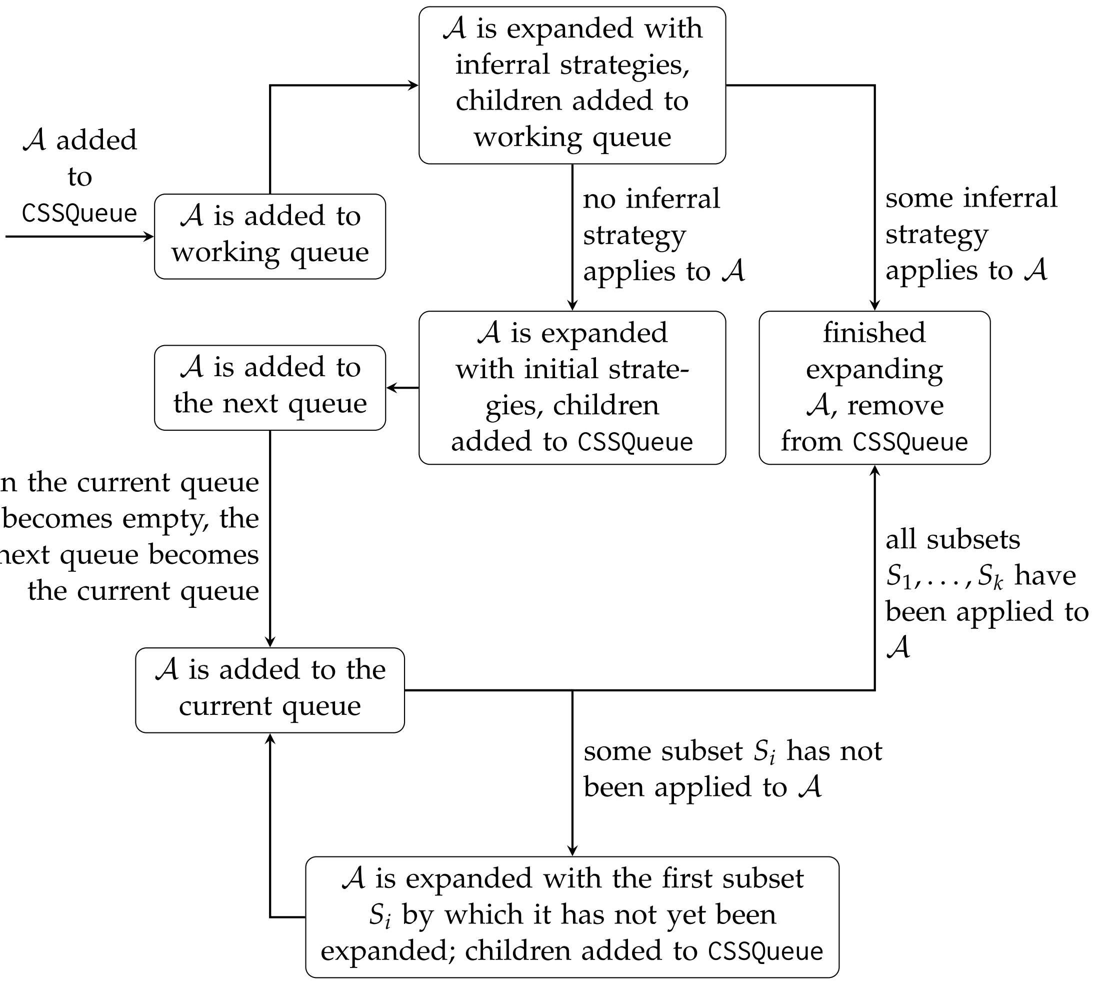

$\mathcal{A}$ added to CSSQueue
$\mathcal{A}$ is added to working queue
$\mathcal{A}$ is expanded with inferral strategies, children added to working queue
no inferral strategy applies to $\mathcal{A}$
some inferral strategy applies to $\mathcal{A}$
$\mathcal{A}$ is expanded with initial strategies, children added to CSSQueue
$\mathcal{A}$ is added to the next queue
finished expanding $\mathcal{A}$, remove from CSSQueue
when the current queue becomes empty, the next queue becomes the current queue
all subsets $S_1, \ldots, S_k$ have been applied to $\mathcal{A}$
$\mathcal{A}$ is added to the current queue
some subset S_i has not been applied to $\mathcal{A}$
$\mathcal{A}$ is expanded with the first subset S_i by which it has not yet been expanded; children added to CSSQueue

Figure 38: The lifecycle of a combinatorial set $\mathcal{A}$ in the default `CSSQueue` employed by `CombSpecSearcher`

using the strategies in the pack designated as expansion strategies. The expansion strategies are further split into subsets of strategies S_1, S_2, ..., S_k. The first time a combinatorial set is taken from the current queue it is expanded using the strategies in set S_1 and added back to the current queue, the second time using the strategies in S_2 and so on. Once the last subset S_k has been applied, we discard the combinatorial set from `CSSQueue` as it has been fully expanded. This allows fine-grained control over how exploration of the universe of sets proceeds simply by modifying the strategy pack. This can be useful when performing Combinatorial Exploration on a set for which one has some intuition about which strategies are more likely to lead to effective decompositions and which are not.

Finally, when both the working queue and the current queue are empty, all sets in the next queue are moved to the current queue, and expansion continues. The lifecycle of an individual combinatorial set $\mathcal{A}$ is outlined in Figure 38.

Every time a combinatorial set $\mathcal{A}$ is discovered for the first time it is assigned a unique integer label and stored in the `ClassDB`. In practice there can be millions of combinatorial sets discovered in a single run of the `CombSpecSearcher`, which imposes a significant memory burden. To mitigate this, our implementation includes the option of storing a compressed version of the combinatorial set, and the `ClassDB` manages the corresponding compression and decompression as needed.

Recall that Subsection 4.3 describes how equivalence strategies may be used to determine that two combinatorial sets have the same enumeration. As such rules violate the productivity conditions, sets are collected into equivalence classes. The `EquivDB` is a union-find data structure that efficiently represents the equivalence classes of combinatorial sets. It contains routines for efficiently merging two equivalence classes, finding a representative for a given equivalence class, and finding a shortest path of equivalence rules between two sets in the same equivalence class.

Immediately after a combinatorial set $\mathcal{A}$ is assigned a label by the `ClassDB`, the `CombSpecSearcher` checks if any of the verification strategies in the strategy pack apply to $\mathcal{A}$. If so, this information is passed to the `EquivDB` because all equivalent classes may also be considered verified.

To expand a combinatorial set $\mathcal{A}$ with a strategy S, the decomposition strategy d_S is applied to $\mathcal{A}$. If S successfully applies to $\mathcal{A}$, it creates a rule $\mathcal{A} \overset{S}{\leftarrow} (\mathcal{B}^{(1)}, \ldots, \mathcal{B}^{(k)})$. Each of the child sets' assigned labels are placed into the `CSSQueue` as described above. The rule is then stored as a tuple $(a, (b_1, \ldots, b_k), S)$ in the `RuleDB`, where a and the b_i are the integer labels assigned to $\mathcal{A}$ and the $\mathcal{B}_i$ and S is the strategy that produced the rule.

Searching for a combinatorial specification is done on the level of equivalence classes of the equivalence relation $\mathcal{R}$ discussed in Subsection 4.3. This task is performed by the `SpecFinder`, which first asks the `RuleDB` for all of the rules found so far. These rules on sets are converted into rules on the equivalence classes of $\mathcal{R}$ by consulting the `EquivDB`. Finally Algorithm 1 from Subsection 3.5 is used to decide if a specification exists. If so, Algorithm 2 chooses one. The items in this specification are equivalence classes of integer labels, and so the `RuleDB`, `EquivDB`, and `ClassDB` work together to convert this back into a specification on combinatorial sets.

The advantage of following the separation of concerns design principle is that the behavior of any one of these components can be easily altered without interfering with the work done by the others. For example, in Subsection 7.4 we briefly introduced the notion of a combinatorial forest, a generalization of a combinatorial specification. The `CombSpecSearcher` can be configured to search instead for combinatorial forests by simply implementing a new `RuleDB` object that has the same functions, but whose internal logic is different. Another example is found in Árdal's thesis [13], in which an alternative `CSSQueue` based on the proof-number search algorithm commonly used on game trees is used to replace our breadth-first approach.

11. Final Notes

Despite the length of this work, we have only scratched the surface of what Combinatorial Exploration can do. While the example domains presented here have all involved sets described by the occurrence or avoidance of certain kinds of patterns, that is just an artifact of our own experience and interests, and is not a requirement for Combinatorial Exploration to apply.

We have mentioned throughout a number of extensions and generalizations that will lead to the automatic discovery of combinatorial specifications for even more combinatorial sets. Some of these have already been completed, or are in progress, and will be discussed in forthcoming work. Others would benefit from the attention of experts in fields other than ours. We will briefly summarize these ideas here and mention a few additional ones as well.

1. In Section 7, we enumerated the 132-avoiding alternating sign matrices by discovering a *combinatorial forest*, which is a set of rules that does not constitute a combinatorial specification because there may be sets on the right-hand sides of rules that appear more than once or not at all on the left-hand sides of rules. Nonetheless, they carry sufficient structural information to enumerate all of the combinatorial sets involved. Although examples of these have appeared sporadically in the literature, to our knowledge they have never been formally defined nor systematically studied. We have devised an algorithmic approach that allows us to efficiently search a set of rules for a combinatorial forest. In our experience, they tend to lead to clever structural descriptions that would be challenging for a human to discover by hand.

   This is already implemented in the Combinatorial Exploration code base [22], and we have used it to find a combinatorial specification for $\mathrm{Av}(1342)$. This class was first enumerated by Bóna [38] by constructing a bijection between it and a set of non-permutation objects; we believe ours is the first direct enumeration. Future work will explore the concept more, lay theoretical foundations, and demonstrate how the use of forests makes Combinatorial Exploration even more effective.

2. In several places, we have mentioned combinatorial specifications that use strategies that lead to extra indices in the counting formulas and "catalytic" variables in systems of equations, e.g., fusion for permutation patterns, and the strategy that merges two identical rows for polyominoes. In order to fully understand these results, the formalized versions of strategies and specifications developed in this work must be extended to multivariate versions, and the notions of reliance graphs and productivity must be suitably adapted.

3. Section 5 discussed that fast uniform random sampling can often be performed when a combinatorial specification has been discovered. Like the other transfer tools discussed, the strategies involved must have certain properties for this to be possible.

4. Suppose that one has combinatorial specifications S_1 and S_2 for two different combinatorial sets $\mathcal{A}$ and $\mathcal{B}$ that may be from different domains. If S_1 and S_2 are structurally similar in the sense that they employ strategies with the same counting functions in precisely the same ways, we call S_1 and S_2 *parallel specifications*. From any two parallel specifications we automatically obtain a size-preserving bijection between $\mathcal{A}$ and $\mathcal{B}$. This work can be found in Eliasson's thesis [75].

5. Our heavy formalization of strategies and specifications may make it possible to employ a theorem verification system such as Lean [110] or Agda [1] to formally verify combinatorial specifications and their enumerations.

6. While we have given short proofs-of-concept for three domains—alternating sign matrices, polyominoes, and set partitions—true success in applying Combinatorial Exploration in these domains is likely to require expertise from researchers who have more experience with these objects than we do. We also believe that Combinatorial Exploration would be effective in other domains such as inversion sequences, ascent sequences, and more.

Acknowledgments

We would like to thank the many undergraduate and masters students with whom we had invaluable discussions and who contributed to this work, including (with references to their theses when applicable): Annija Apine, Ragnar Páll Árdal [13], Arnar Bjarni Arnarsson [14, 15], Alfur Birkir Bjarnason [15], Jon Steinn Eliasson [75], Unnar Freyr Erlendsson [15, 76], Kolbeinn Páll Erlingsson [87], Bjarki Ágúst Guðmundsson, Björn Gunnarsson [87], Sigurður Helgason [126], Kristmundur Ágúst Jónsson [87], Tómas Ken Magnússon [111], James Robb [126], Óðinn Hjaltason Schiöth, Murray Tannock, and Sigurjón Freyr Viktorsson [15].

We'd also like to thank Vince Vatter for providing improved code for processing our sampled data into heatmap images. We are further grateful to the referees for their suggestions, which have improved this work.

CB's research is supported by grant 207178-052 from the Icelandic Research Fund.

ÉN's research is supported by the Reykjavik University Research Fund.

JP's research is supported by grant 713579 from the Simons Foundation.

References

[1] *Agda*. URL: `https://github.com/agda/agda`.

[2] Michael H. Albert, M. D. Atkinson, and Robert Brignall. The enumeration of permutations avoiding 2143 and 4231. *Pure Math. Appl. (PU.M.A.)* 22.2 (2011), pp. 87–98.

[3] Michael H. Albert, M. D. Atkinson, and Robert Brignall. The enumeration of three pattern classes using monotone grid classes. *Electron. J. Combin.* 19.3 (2012), Paper 20, 34.

[4] Michael H. Albert, M. D. Atkinson, and Vincent Vatter. Counting 1324,4231-avoiding permutations. *Electron. J. Combin.* (2009), pp. 1–9.

[5] Michael H. Albert, M. D. Atkinson, and Vincent Vatter. Inflations of geometric grid classes: three case studies. *Australas. J. Combin.* 58.1 (2014), pp. 27–47.

[6] Michael H. Albert, Christian Bean, Anders Claesson, Émile Nadeau, Jay Pantone, and Henning Ulfarsson. *The Permutation Pattern Avoidance Library (PermPAL)*. URL: `https://permpal.com`.

[7] Michael H. Albert and Robert Brignall. Enumerating indices of Schubert varieties defined by inclusions. *Journal of Combinatorial Theory, Series A* 123.1 (2014), pp. 154–168.

[8] Michael H. Albert, M. Elder, a. Rechnitzer, P. Westcott, and M. Zabrocki. On the Stanley–Wilf limit of 4231-avoiding permutations and a conjecture of Arratia. *Adv. Appl. Math.* 36.2 (2006), pp. 96–105.

[9] Michael H. Albert, Cheyne Homberger, Jay Pantone, Nathaniel Shar, and Vincent Vatter. Generating permutations with restricted containers. *J. Combin. Theory Ser. A* 157 (2018), pp. 205–232.

[10] Michael H. Albert, Jay Pantone, and Vincent Vatter. On the growth of merges and staircases of permutation classes. *Rocky Mountain J. Math.* 49.2 (2019), pp. 355–367.

[11] Gadi Aleksandrowicz, Andrei Asinowski, and Gill Barequet. A polyominoes-permutations injection and tree-like convex polyominoes. *J. Combin. Theory Ser. A* 119.3 (2012), pp. 503–520.

[12] Timothy Alland and Edward Richmond. Pattern avoidance and fiber bundle structures on Schubert varieties. *J. Combin. Theory Ser. A* 154 (2018), pp. 533–550.

[13] Ragnar Páll Árdal. "Proof-number search in automated enumeration of combinatorial objects". Reykjavik University, 2022. URL: http://hdl.handle.net/1946/40466.

[14] Arnar Bjarni Arnarson. "Substitution decomposition for permutation classes with infinitely many simple permutations". Reykjavik University, 2019. URL: http://hdl.handle.net/1946/33583.

[15] Arnar Bjarni Arnarson, Álfur Birkir Bjarnason, Sigurjón Freyr Viktorsson, and Unnar Freyr Erlendsson. "PermPAL - Permutation pattern avoidance library". Reykjavik University, 2017. URL: http://hdl.handle.net/1946/28794.

[16] M. D. Atkinson. Permutations which are the union of an increasing and a decreasing subsequence. *Electron. J. Combin.* 5 (1998), Research paper 6, 13.

[17] M. D. Atkinson, N. Ruškuc, and Rebecca Smith. Substitution-closed pattern classes. *J. Combin. Theory Ser. A* 118.2 (2011), pp. 317–340.

[18] M. D. Atkinson, Bruce E. Sagan, and Vincent Vatter. Counting (3+1)-avoiding permutations. *European J. Combin.* 33.1 (2012), pp. 49–61.

[19] Elena Barcucci, Alberto Del Lungo, Elisa Pergola, and Renzo Pinzani. ECO: a methodology for the enumeration of combinatorial objects. *Journal of Difference Equations and Applications* 5.4-5 (1999), pp. 435–490.

[20] Frédérique Bassino, Mathilde Bouvel, Adeline Pierrot, Carine Pivoteau, and Dominique Rossin. An algorithm computing combinatorial specifications of permutation classes. *Discrete Appl. Math.* 224 (2017), pp. 16–44.

[21] Christian Bean, Jon Steinn Eliasson, Tomas Ken Magnusson, Émile Nadeau, Jay Pantone, and Henning Ulfarsson. *Tilings*. Version 3.0.0. 2021. DOI: 10.5281/zenodo.4948344. URL: https://github.com/PermutaTriangle/Tilings.

[22] Christian Bean, Jon Steinn Eliasson, Émile Nadeau, Jay Pantone, and Henning Ulfarsson. *Comb_spec_searcher*. Version 4.0.0. 2021. DOI: 10.5281/zenodo.4946832. URL: https://github.com/PermutaTriangle/comb_spec_searcher.

[23] Christian Bean, Bjarki Gudmundsson, and Henning Ulfarsson. Automatic discovery of structural rules of permutation classes. *Math. Comp.* 88.318 (2019), pp. 1967–1990.

[24] François. Bergeron, Gilbert Labelle, and Pierre Leroux. *Combinatorial species and tree-like structures*. Vol. 67. Encyclopedia of Mathematics and its Applications. Cambridge University Press, Cambridge, 1998, pp. xx+457.

[25] David Bevan. Permutations avoiding 1324 and patterns in Łukasiewicz paths. *J. London Math. Soc.* 92.1 (2015), pp. 105–122.

[26] David Bevan. The permutation class Av(4213,2143). *Discrete Math. Theor. Comput. Sci.* 18.2 (2016), p. 14.

[27] David Bevan. The permutation classes $\mathrm{Av}(1234, 2341)$ and $\mathrm{Av}(1243, 2314)$. *Australas. J. Comb.* 64 (2016), pp. 3–20.

[28] David Bevan, Robert Brignall, Andrew Elvey Price, and Jay Pantone. A structural characterisation of Av(1324) and new bounds on its growth rate. *European J. Combin.* (2020), p. 103115.

[29] David Bevan, Robert Brignall, Andrew Elvey Price, and Jay Pantone. Staircases, dominoes, and the growth rate of 1324-avoiders. *Electron. Notes Discrete Math.* 61 (2017), pp. 123–129.

[30] Yonah Biers-Ariel. *Flexible Schemes and Beyond: Experimental Enumeration of Pattern Avoidance Classes*. ProQuest LLC, Ann Arbor, MI, 2020, p. 62.

[31] Yonah Biers-Ariel, Haripriya Chakraborty, John Chiarelli, Brian Ek, Andrew Lohr, Jinyoung Park, Justin Semonsen, Richard Voepel, Mingjia Yang, Anthony Zaleski, and Doron Zeilberger. *Counting Permutations that Avoid Many Patterns*. 2017. arXiv: `1703.02415 [math.CO]`.

[32] Jonathan Bloom and Alexander Burstein. Egge triples and unbalanced Wilf-equivalence. *Australas. J. Combin.* 64 (2016), pp. 232–251.

[33] Jonathan Bloom and Sergi Elizalde. Pattern avoidance in matchings and partitions. *Electron. J. Combin.* 20.2 (2013), Paper 5, 38.

[34] Jonathan Bloom and Dan Saracino. Pattern avoidance for set partitions á la Klazar. *Discrete Math. Theor. Comput. Sci.* 18.2 (2016), Paper No. 9, 22.

[35] Jonathan Bloom and Vincent Vatter. Two vignettes on full rook placements. *Australas. J. Comb.* 64 (2016), pp. 77–87.

[36] Miklós Bóna. A new record for 1324-avoiding permutations. *European J. Combin.* 1.1 (2015), pp. 198–206.

[37] Miklós Bóna. A new upper bound for 1324-avoiding permutations. *Combin. Probab. Comput.* 23.5 (2014), pp. 717–724.

[38] Miklós 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.

[39] Miklós Bóna. The permutation classes equinumerous to the smooth class. *Electron. J. Combin.* 5 (1998), Research Paper 31, 12.

[40] Miklós Bóna and Jay Pantone. Permutations avoiding sets of patterns with long monotone subsequences. *J. Symbolic Comput.* 116 (2023), pp. 130–138.

[41] Mireille Bousquet-Mélou. A method for the enumeration of various classes of column-convex polygons. *Discrete Mathematics* 154.1-3 (1996), pp. 1–25.

[42] Mireille Bousquet-Mélou. "Four classes of pattern-avoiding permutations under one roof: generating trees with two labels". In: vol. 9. 2. 2002/03, Research paper 19, 31.

[43] Mireille Bousquet-Mélou and Jean-Marc Fédou. The generating function of convex polyominoes: The resolution of a q-differential system. *Discrete Mathematics* 137.1-3 (1995), pp. 53–75.

[44] Mireille Bousquet-Mélou and Arnaud Jehanne. Polynomial equations with one catalytic variable, algebraic series and map enumeration. *J. Combin. Theory Ser. B* 96.5 (2006), pp. 623–672.

[45] Robert Brignall, Michael Engen, and Vincent Vatter. A counterexample regarding labelled well-quasi-ordering. *Graphs Combin.* 34.6 (2018), pp. 1395–1409.

[46] Robert Brignall and Vincent Vatter. Labelled well-quasi-order for permutation classes. *Comb. Theory* 2.3 (2022), Paper No. 14, 54.

[47] Simon R. Broadbent and John M. Hammersley. Percolation processes: I. Crystals and mazes. *Mathematical Proceedings of the Cambridge Philosophical Society* 53.3 (1957), pp. 629–641.

[48] Alexander Burstein and Jay Pantone. Two examples of unbalanced Wilf-equivalence. *J. Comb.* 6.1-2 (2015), pp. 55–67.

[49] David Callan. *Permutations avoiding 4321 and 3241 have an algebraic generating function*. 2013. arXiv: `1306.3193 [math.CO]`.

[50] David Callan. *The number of 1243, 2134-avoiding permutations*. 2013. arXiv: `1303.3857 [math.CO]`.

[51] David Callan and Toufik Mansour. Enumeration of 2-wilf classes of four 4-letter patterns. *Turkish Journal of Analysis and Number Theory* 5.6 (2017), pp. 210–225.

[52] David Callan and Toufik Mansour. Enumeration of 3- and 4-Wilf classes of four 4-letter patterns. *Notes on Number Theory and Discrete Mathematics* 24.3 (2018), pp. 115–130.

[53] David Callan and Toufik Mansour. *Enumeration of small Wilf classes avoiding 1324 and two other 4-letter patterns*. 2017. arXiv: `1705.00933 [math.CO]`.

[54] David Callan and Toufik Mansour. *Enumeration of small Wilf classes avoiding 1342 and two other 4-letter patterns*. 2017. arXiv: `1708.00832 [math.CO]`.

[55] David Callan and Toufik Mansour. On permutations avoiding 1243, 2134, and another 4-letter pattern. *Pure Math. Appl. (PU.M.A.)* 26.1 (2017), pp. 11–21.

[56] David Callan and Toufik Mansour. On permutations avoiding 1324, 2143, and another 4-letter pattern. *Pure Math. Appl. (PU.M.A.)* 26.1 (2017), pp. 1–10.

[57] David Callan, Toufik Mansour, and Mark Shattuck. *Enumeration of permutations avoiding a triple of 4-letter patterns is all done*. 2017. arXiv: `1709.04279 [math.CO]`.

[58] David Callan, Toufik Mansour, and Mark Shattuck. Wilf classification of triples of 4-letter patterns I. *Discrete Math. Theor. Comput. Sci.* 19.1 (2017), Paper No. 5, 35.

[59] David Callan, Toufik Mansour, and Mark Shattuck. Wilf classification of triples of 4-letter patterns II. *Discrete Math. Theor. Comput. Sci.* 19.1 (2017), Paper No. 6, 44.

[60] Lindsey R. Campbell, Samantha Dahlberg, Robert Dorward, Jonathan Gerhard, Thomas Grubb, Carlin Purcell, and Bruce E. Sagan. Restricted growth function patterns and statistics. *Adv. in Appl. Math.* 100 (2018), pp. 1–42.

[61] Giusi Castiglione, Andrea Frosini, Antonio Restivo, and Simone Rinaldi. Enumeration of L-convex polyominoes by rows and columns. *Theoretical Computer Science* 347.1-2 (2005), pp. 336–352.

[62] William Y. C. Chen, Eva Y. P. Deng, Rosena R. X. Du, Richard P. Stanley, and Catherine H. Yan. Crossings and nestings of matchings and partitions. *Trans. Amer. Math. Soc.* 359.4 (2007), pp. 1555–1575.

[63] Fan R. K. Chung, Ronald L. Graham, Verner E. Hoggatt Jr., and Mark Kleiman. The number of Baxter permutations. *J. Combin. Theory Ser. A* 24.3 (1978), pp. 382–394.

[64] Anders Claesson, Vít Jelínek, and Einar Steingrímsson. Upper bounds for the Stanley–Wilf limit of 1324 and other layered patterns. *J. Combin. Theory Ser. A* 119.8 (2012), pp. 1680–1691.

[65] Andrew R. Conway and Anthony J Guttmann. On two-dimensional percolation. *Journal of Physics A: Mathematical and General* 28.4 (1995), pp. 891–904.

[66] Andrew R. Conway and Anthony J. Guttmann. On 1324-avoiding permutations. *Adv. in Appl. Math.* 64 (2015), pp. 50–69.

[67] Andrew R. Conway, Anthony J. Guttmann, and Paul Zinn-Justin. 1324-avoiding permutations revisited. *Advances in Applied Mathematics* 96 (2018), pp. 312–333.

[68] Samantha Dahlberg, Robert Dorward, Jonathan Gerhard, Thomas Grubb, Carlin Purcell, Lindsey Reppuhn, and Bruce E. Sagan. Set partition patterns and statistics. *Discrete Math.* 339.1 (2016), pp. 1–16.

[69] Colin Defant. Stack-sorting preimages of permutation classes. *Sém. Lothar. Combin.* 82 (2020), Art. B82b, 41.

[70] Deepak Dhar. Exact solution of a directed-site animals-enumeration problem in three dimensions. *Phys. Rev. Lett.* 51 (1983), pp. 853–856.

[71] Stoyan Dimitrov. On permutation patterns with constrained gap sizes. *Australas. J. Combin.* 87 (2023), pp. 98–128.

[72] Eric Egge. *Some new pattern-avoiding permutations counted by the Schröder numbers, talk, AMS Fall Eastern Meeting, Rochester, NY, September 2012*.

[73] Murray Elder. Permutations generated by a stack of depth 2 and an infinite stack in series. *Electron. J. Combin.* 13.1 (2006), Research Paper 68, 12.

[74] Murray Elder, Geoffrey Lee, and Andrew Rechnitzer. Permutations generated by a depth 2 stack and an infinite stack in series are algebraic. *Electron. J. Combin.* 22.2 (2015), Paper 2.16, 23.

[75] Jon Steinn Eliasson. "Automated bijections with Combinatorial Exploration". Reykjavik University, 2022. URL: `http://hdl.handle.net/1946/40459`.

[76] Unnar Freyr Erlendsson. "Effective enumeration of permutation classes and their juxtapositions". Reykjavik University, 2019. URL: `http://hdl.handle.net/1946/33586`.

[77] Jean-Marc Fedou, Simone Rinaldi, and Andrea Frosini. Enumeration of 4-stack polyominoes. *Theoretical Computer Science* 502 (2013), pp. 88–97.

[78] Alex Fink, Karola Mészáros, and Avery St. Dizier. Zero-one Schubert polynomials. *Math. Z.* 297.3-4 (2021), pp. 1023–1042.

[79] Ghassan Firro and Toufik Mansour. Three-letter-pattern-avoiding permutations and functional equations. *Electron. J. Combin.* 13.1 (2006), Research Paper 51, 14.

[80] Ilse Fischer. A new proof of the refined alternating sign matrix theorem. *J. Combin. Theory Ser. A* 114.2 (2007), pp. 253–264.

[81] Philippe Flajolet and Robert Sedgewick. *Analytic combinatorics*. Cambridge: Cambridge University Press, 2009, pp. xiv+810.

[82] Andrea Frosini, Veronica Guerrini, and Simone Rinaldi. Geometric properties of matrices induced by pattern avoidance. *Theoret. Comput. Sci.* 624 (2016), pp. 109–120.

[83] Alice L. L. Gao and Sergey Kitaev. On partially ordered patterns of lengths 4 and 5 in permutations. *Electron. J. Combin.* 26.3 (2019), Paper No. 3.26, 31.

[84] Vesselin Gasharov and Victor Reiner. Cohomology of smooth Schubert varieties in partial flag manifolds. *J. London Math. Soc. (2)* 66.3 (2002), pp. 550–562.

[85] Ira M. Gessel. Symmetric functions and p-recursiveness. *J. Combin. Theory Ser. A* 53.2 (1990), pp. 257–285.

[86] Alain Goupil, Hugo Cloutier, and Fathallah Nouboud. Enumeration of polyominoes inscribed in a rectangle. *Discrete Appl. Math.* 158.18 (2010), pp. 2014–2023.

[87] Björn Gunnarsson, Kolbeinn Páll Erlingsson, and Kristmundur Ágúst Jónsson. "Identifying structures in Motzkin paths". Reykjavik University, 2019. URL: `http://hdl.handle.net/1946/32304`.

[88] Anthony J. Guttmann and Václav Kotěšovec. A numerical study of L-convex polyominoes and 201-avoiding ascent sequences. *Sém. Lothar. Combin.* 87B (2023), Art. 2, 11.

[89] Christopher Hoffman, Douglas Rizzolo, and Erik Slivken. Pattern-avoiding permutations and brownian excursion part I: shapes and fluctuations. *Random Struct. Algorithms* 50.3 (2017), pp. 394–419.

[90] Cheyne Homberger and Vincent Vatter. On the effective and automatic enumeration of polynomial permutation classes. *J. Symbolic Comput.* 76 (2016), pp. 84–96.

[91] Masaki Ikeda. *Enumeration of permutations indexing local complete intersection Schubert varieties*. ProQuest LLC, Ann Arbor, MI, 2016, p. 221.

[92] E. J. Janse van Rensburg. *The statistical mechanics of interacting walks, polygons, animals and vesicles*. 2nd. Oxford Lecture Series in Mathematics and Its Applications, 2015.

[93] Vít Jelínek, Toufik Mansour, and Mark Shattuck. On multiple pattern avoiding set partitions. *Adv. in Appl. Math.* 50.2 (2013), pp. 292–326.

[94] Iwan Jensen. "Counting polyominoes: a parallel implementation for cluster computing". In: *Computational Science — ICCS 2003*. Ed. by Peter M. A. Sloot, David Abramson, Alexander V. Bogdanov, Yuriy E. Gorbachev, Jack J. Dongarra, and Albert Y. Zomaya. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 203–212.

[95] Fredrik Johansson and Brian Nakamura. Using functional equations to enumerate 1324-avoiding permutations. *Adv. Appl. Math.* 56 (2014), pp. 20–34.

[96] Robert Johansson and Svante Linusson. Pattern avoidance in alternating sign matrices. *Ann. Comb.* 11.3-4 (2007), pp. 471–480.

[97] André Joyal. "Foncteurs analytiques et espèces de structures". In: *Combinatoire énumérative (Montreal, Que., 1985/Quebec, Que., 1985)*. Vol. 1234. Lecture Notes in Math. Springer, Berlin, 1986, pp. 126–159.

[98] André Joyal. Une théorie combinatoire des séries formelles. *Adv. in Math.* 42.1 (1981), pp. 1–82.

[99] David A. Klarner. Cell growth problems. *Canadian J. Math.* 19 (1967), pp. 851–863.

[100] Martin Klazar. Counting pattern-free set partitions. I. A generalization of Stirling numbers of the second kind. *European J. Combin.* 21.3 (2000), pp. 367–378.

[101] Martin Klazar. Counting pattern-free set partitions. II. Noncrossing and other hypergraphs. *Electron. J. Combin.* 7 (2000), Research Paper 34, 25.

[102] Martin Klazar. On *abab*-free and *abba*-free set partitions. *European J. Combin.* 17.1 (1996), pp. 53–68.

[103] Aaron J. Klein, Joel Brewster Lewis, and Alejandro H. Morales. Counting matrices over finite fields with support on skew Young diagrams and complements of Rothe diagrams. *J. Algebraic Combin.* 39.2 (2014), pp. 429–456.

[104] Donald E. Knuth. *The art of computer programming. Vol. 1: Fundamental algorithms*. Addison-Wesley Publishing Co., Reading, Mass.-London-Don Mills, Ont, 1969, pp. xxi+634.

[105] Darla Kremer. Permutations with forbidden subsequences and a generalized schroder number. *Discrete Math.* 218.1-3 (2000), pp. 121–130.

[106] Darla Kremer. Postscript: "Permutations with forbidden subsequences and a generalized Schröder number". *Discrete Math.* 270.1-3 (2003), pp. 333–334.

[107] Darla Kremer and Wai Chee Shiu. Finite transition matrices for permutations avoiding pairs of length four patterns. *Discrete Math.* 268.1-3 (2003), pp. 171–183.

[108] Greg Kuperberg. Another proof of the alternating-sign matrix conjecture. *Internat. Math. Res. Notices* 3 (1996), pp. 139–150.

[109] Ian Le. Wilf classes of pairs of permutations of length 4. *Electron. J. Combin.* 12 (2005), Paper 25.

[110] *Lean*. URL: `https://leanprover.github.io/`.

[111] Tómas Ken Magnússon. "Forced permutation patterns and applications to coincidence classification of mesh patterns and enumeration of permutation classes". Reykjavik University, 2018. URL: `http://hdl.handle.net/1946/29902`.

[112] Toufik Mansour. Enumeration and Wilf-classification of permutations avoiding five patterns of length 4. *Contrib. Math.* 1 (2020), pp. 1–10.

[113] Toufik Mansour. Enumeration and Wilf-classification of permutations avoiding four patterns of length 4. *DML, Discrete Math. Lett.* 3 (2020), pp. 67–94.

[114] Toufik Mansour and Matthias Schork. Wilf classification of subsets of eight and nine four-letter patterns. *J. Comb. Number Theory* 8.3 (2016), pp. 257–283.

[115] Toufik Mansour and Matthias Schork. Wilf classification of subsets of four letter patterns. *J. Comb. Number Theory* 8.1 (2016), pp. 1–129.

[116] Toufik Mansour and Matthias Schork. Wilf classification of subsets of six and seven four-letter patterns. *J. Comb. Number Theory* 9.3 (2017), pp. 169–213.

[117] Darko Marinov and Radoš Radoičić. Counting 1324-avoiding permutations. *Electron. J. Combin.* 9.2 (2003), Research paper 13, 9.

[118] Sam Miner. *Enumeration of several two-by-four classes*. arXiv: 1610.01908 [math.CO].

[119] Sam Miner and Igor Pak. The shape of random pattern-avoiding permutations. *Adv. Appl. Math.* 55 (2014), pp. 86–130.

[120] Samuel Miner and Jay Pantone. *Completing the structural analysis of the 2x4 permutation classes*. arXiv: 1802.00483 [math.CO].

[121] Maximillian M. Murphy and Vincent R. Vatter. "Profile classes and partial well-order for permutations". In: vol. 9. 2. 2002, Research paper 17, 30.

[122] Jay Pantone. The enumeration of permutations avoiding 3124 and 4312. *Ann. Comb.* 21.2 (2017), pp. 293–315.

[123] Carine Pivoteau, Bruno Salvy, and Michèle Soria. Algorithms for combinatorial structures: well-founded systems and Newton iterations. *J. Combin. Theory Ser. A* 119.8 (2012), pp. 1711–1773.

[124] H. N. de Ridder et al. *Information System on Graph Classes and their Inclusions (ISGCI)*. URL: http://www.graphclasses.org.

[125] John Riordan. The distribution of crossings of chords joining pairs of $2n$ points on a circle. *Math. Comp.* 29 (1975), pp. 215–222.

[126] James Robb and Sigurður Helgason. "Identifying combinatorial structures for binary strings and set partitions". Reykjavik University, 2018. URL: http://hdl.handle.net/1946/31176.

[127] Bruce E. Sagan. Pattern avoidance in set partitions. *Ars Combin.* 94 (2010), pp. 79–96.

[128] Neil J. A. Sloane. *The Online Encyclopedia of Integer Sequences*. URL: http://oeis.org.

[129] Zvezdelina Stankova. Classification of forbidden subsequences of length 4. *European J. Combin.* 17.5 (1996), pp. 501–517.

[130] Zvezdelina Stankova. Forbidden subsequences. *Discrete Math.* 132.1-3 (1994), pp. 291–316.

[131] Bridget Eileen Tenner. *Database of Permutation Pattern Avoidance*. URL: https://math.depaul.edu/bridget/patterns.html.

[132] Jacques Touchard. Sur un problème de configurations. *C. R. Acad. Sci. Paris* 230 (1950), pp. 1997–1998.

[133] Henning Ulfarsson and Alexander Woo. Which Schubert varieties are local complete intersections? *Proc. Lond. Math. Soc. (3)* 107.5 (2013), pp. 1004–1052.

[134] Vincent Vatter. Enumeration schemes for restricted permutations. *Combin. Probab. Comput.* 17.1 (2008), pp. 137–159.

[135] Vincent Vatter. Finding regular insertion encodings for permutation classes. *J. Symbolic Comput.* 47.3 (2012), pp. 259–265.

[136] Vincent Vatter. Finitely labeled generating trees and restricted permutations. *J. Symbolic Comput.* 41.5 (2006), pp. 559–572.

[137] Vincent Vatter. "Permutation classes". In: *Handbook of Combinatorics*. Ed. by Miklós Bóna. CRC Press, 2015.

[138] Julian West. Generating trees and the Catalan and Schröder numbers. *Discrete Math.* 146.1-3 (1995), pp. 247–262.

[139] Doron Zeilberger. Enumeration schemes and, more importantly, their automatic generation. *Ann. Comb.* 2.2 (1998), pp. 185–195.

[140] Doron Zeilberger. "Proof of the alternating sign matrix conjecture". In: vol. 3. 2. 1996, Research Paper 13, approx. 84.

[141] Doron Zeilberger. The umbral transfer-matrix method. I. Foundations. *Journal of Combinatorial Theory, Series A* 91.1 (2000), pp. 451–463.