Pattern-Avoiding Binary Trees – Generation, Counting, and Bijections

Petr Gregor
Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic

Torsten Mütze
Department of Computer Science, University of Warwick, United Kingdom
Department of Theoretical Computer Science and Mathematical Logic,
Charles University, Prague, Czech Republic

Namrata
Department of Computer Science, University of Warwick, Coventry, UK

Abstract

In this paper we propose a notion of pattern avoidance in binary trees that generalizes the avoidance of contiguous tree patterns studied by Rowland and non-contiguous tree patterns studied by Dairyko, Pudwell, Tyner, and Wynn. Specifically, we propose algorithms for generating different classes of binary trees that are characterized by avoiding one or more of these generalized patterns. This is achieved by applying the recent Hartung–Hoang–Mütze–Williams generation framework, by encoding binary trees via permutations. In particular, we establish a one-to-one correspondence between tree patterns and certain mesh permutation patterns. We also conduct a systematic investigation of all tree patterns on at most 5 vertices, and we establish bijections between pattern-avoiding binary trees and other combinatorial objects, in particular pattern-avoiding lattice paths and set partitions.

2012 ACM Subject Classification Mathematics of computing → Enumeration; Mathematics of computing → Permutations and combinations; Mathematics of computing → Combinatorial algorithms

Keywords and phrases Generation, binary tree, pattern avoidance, permutation, bijection

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2023.33

Related Version *Full Version*: `https://arxiv.org/abs/2306.08420` [20]

Funding This work was supported by Czech Science Foundation grant GA 22-15272S.

1 Introduction

Pattern avoidance is a central theme in combinatorics and discrete mathematics. For example, in Ramsey theory one investigates how order arises in large unordered structures such as graphs, hypergraphs, or subsets of the integers. The concept also arises naturally in algorithmic applications. For example, Knuth [28] showed that the integer sequences that are sortable by one pass through a stack are precisely 231-avoiding permutations. Pattern-avoiding permutations are a particularly important and heavily studied strand of research, one that comes with its own associated conference "Permutation Patterns", held annually since 2003. While it may seem that pattern-avoiding permutations are somewhat limited in scope, via suitable bijections they actually encode many objects studied in other branches of combinatorics. Pattern avoidance has also been studied directly in these other classes of objects, such as trees [38, 13, 12, 11, 15, 37, 6, 1, 16], set partitions [29, 24, 25, 26, 18, 22, 32, 33, 39, 19, 23, 17, 8], lattice paths [40, 5, 2, 4], heaps [30], matchings [7], and rectangulations [34]. In this work, we focus on binary trees, a class of objects that is fundamental within computer science, and also a classical Catalan family.

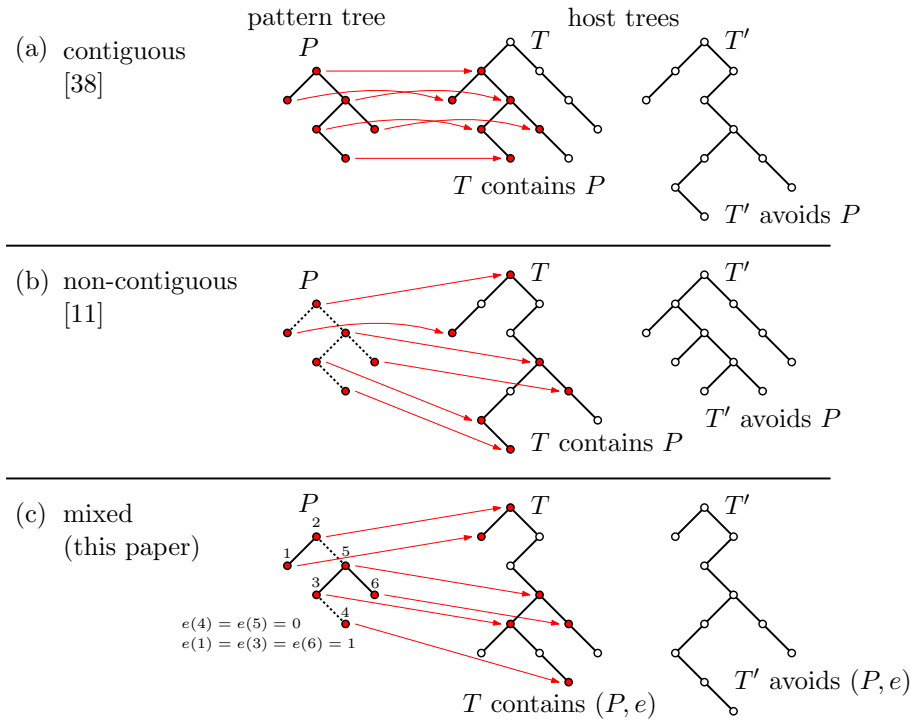


Figure 1 Illustration of different notions of pattern containment in binary trees. Contiguous edges are drawn solid, whereas non-contiguous edges are drawn dotted.

So far, two different notions of pattern avoidance in binary trees have been studied in the literature. We consider a binary tree T, which serves as the host tree, and another binary tree P, which serves as the pattern tree. Rowland [38] considered a *contiguous* notion of pattern containment, where T contains P if P is present as an induced subtree of T; see Figure 1 (a). He devised an algorithm to compute the generating function for the number of n-vertex binary trees that avoid P, and he showed that this generating function is always algebraic. Dairyko, Pudwell, Tyner, and Wynn [11] considered a *non-contiguous* notion of pattern containment, where T contains P if P is present as a "minor" of T; see Figure 1 (b). They discovered the remarkable phenomenon that for any two distinct k-vertex pattern trees P and P', the number of n-vertex host trees that avoid P is the same as the number of trees that avoid P', i.e., P and P' are *Wilf-equivalent* patterns. They also obtain the corresponding generating function (which is independent of P, but only depends on k and n).

In this paper, we consider *mixed* tree patterns, which generalize both of the two aforementioned types of tree patterns, by specifying separately for each edge of P whether it is considered contiguous or non-contiguous, i.e., whether its end vertices in the occurrence of the pattern must be in a parent-child or ancestor-descendant relationship (in the correct direction left/right), respectively; see Figure 1 (c). Observe that the notions of tree patterns considered in [38] and [11] are the tree analogues of consecutive [14] and classical permutation patterns, respectively. Our new notion of mixed patterns is the tree analogue of vincular permutation patterns [3], which generalize classical and consecutive permutation patterns.

1.1 The Lucas–Roelants van Baronaigien–Ruskey algorithm

One of the goals in this paper is to generate different classes of binary trees, i.e., we seek an algorithm that visits every tree from the class exactly once. Our starting point is a classical result due to Lucas, Roelants van Baronaigien, and Ruskey [31], which asserts that all n-vertex binary trees can be generated by tree rotations, i.e., every tree is obtained from its predecessor by a single *tree rotation* operation; see Figures 2 and 3.

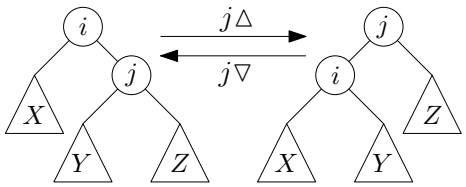


Figure 2 Rotation in binary trees.

The algorithm is an instance of a *combinatorial Gray code* [41, 35], which is a listing of objects such that any two consecutive objects differ in a "small local" change. The aforementioned Gray code algorithm for binary trees can be implemented in time $\mathcal{O}(1)$ per generated tree.

Williams [42] discovered a stunningly simple description of the Lucas–Roelants van Baronaigien–Ruskey Gray code for binary trees via the following greedy algorithm, which is based on labeling the vertices with $1, \ldots, n$ according to the search tree property: Start with the right path, and then repeatedly perform a tree rotation with the largest possible vertex that creates a previously unvisited tree.

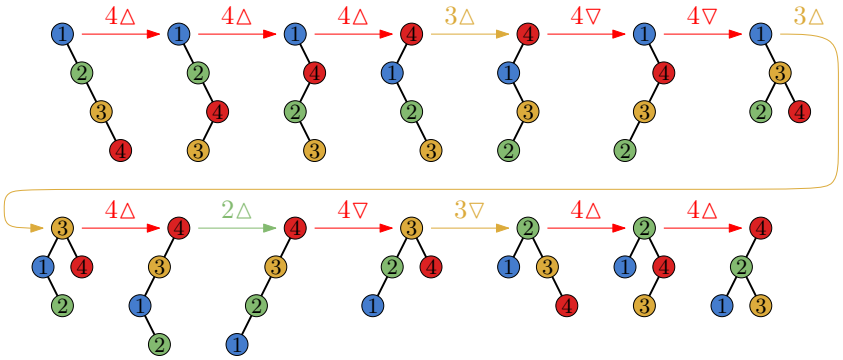


Figure 3 The Lucas–Roelants van Baronaigien–Ruskey algorithm to generate all binary trees with $n = 4$ vertices by tree rotations. The vertices are labeled with $1, 2, 3, 4$ according to the search tree property.

1.2 Our results

It is well known that binary trees are in bijection with 231-avoiding permutations. Our first result generalizes this bijection, by establishing a one-to-one correspondence between mixed binary tree patterns and mesh permutation patterns, a generalization of classical permutation patterns due to Brändén and Claesson [9]. Specifically, we show that n-vertex binary trees that avoid a particular (mixed) tree pattern P are in bijection with 231-avoiding permutations that avoid a corresponding mesh pattern $\sigma(P)$ (see Theorem 2 below).

This bijection enables us to apply the Hartung–Hoang–Mütze–Williams generation framework [21], which is based on permutations. We thus obtain algorithms for efficiently generating different classes of pattern-avoiding binary trees, which work under some mild conditions on the tree pattern(s). These algorithms are all based on a simple greedy algorithm, which generalizes Williams' algorithm for the Lucas–Roelants van Baronaigien–Ruskey Gray code of binary trees (see Algorithm S, Algorithm H, and Theorems 3 and 4, respectively). Specifically, instead of tree rotations our algorithms use a more general operation that we refer to as a *slide*. We implemented our generation algorithm in C++, and we made it available for download and experimentation on the Combinatorial Object Server [10].

For our new notion of mixed tree patterns, we conduct a systematic investigation of all tree patterns on up to 5 vertices. This gives rise to many counting sequences, some already present in the OEIS [36] and some new to it, giving rise to several interesting conjectures. In this work we establish most of these as theorems, by proving bijections between different classes of pattern-avoiding binary trees and other combinatorial objects, in particular pattern-avoiding lattice paths (Section 6 and [20, Sec. 7.2.4]) and set partitions ([20, Thm. 15]).

1.3 Outline of this paper

In Section 2 we introduce notations that will be used throughout the paper. In Section 3 we establish a bijection between binary trees patterns and mesh patterns. In Section 4 we present our algorithms for generating pattern-avoiding binary trees. In Section 5 we report on our computational results for all small tree patterns. In Section 6 we prove bijections between pattern-avoiding binary trees and Motzkin paths. Some open problems are discussed in Section 7. Due to space constraints, this extended abstract omits proofs, and several further results, illustrations and tables; see [20].

2 Preliminaries

In this section we introduce a few general definitions related to binary trees, and we define our notion of pattern avoidance for those objects.

2.1 Binary tree notions

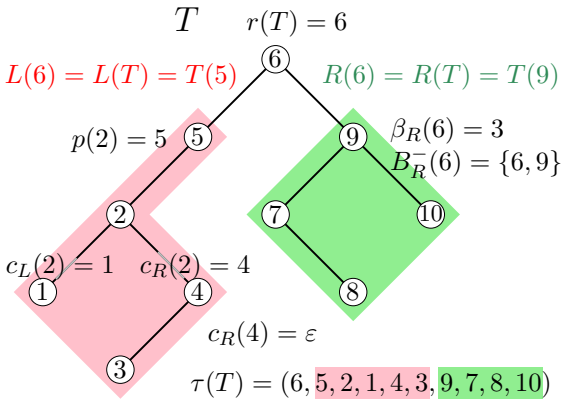


Figure 4 Definitions related to binary trees.

We consider binary trees whose vertex set is a set of consecutive integers $\{i, i+1, \ldots, j\}$. In particular, we write $\mathcal{T}_n$ for the set of binary trees with the vertex set $[n] := \{1, 2, \ldots, n\}$. The vertex labels of each tree are defined uniquely by the *search tree property*, i.e., for any vertex i, all its left descendants are smaller than i and all its right descendants are greater than i. The special empty tree with $n = 0$ vertices is denoted by ε, so $\mathcal{T}_0 = \{\varepsilon\}$. The following definitions are illustrated in Figure 4. For any binary tree T, we denote the root of T by $r(T)$. For any vertex i of T, its left and right child are denoted by $c_L(i)$ and $c_R(i)$, respectively, and its parent is denoted by $p(i)$. If i does not have a left child, a right child or a parent, then we define $c_L(i) := \varepsilon$, $c_R(i) := \varepsilon$, or $p(i) := \varepsilon$, respectively. Furthermore, we write $T(i)$ for the subtree of T rooted at i. Also, we define $L(i) := T(c_L(i))$ if $c_L(i) \neq \varepsilon$ and $L(i) := \varepsilon$ otherwise, and $R(i) := T(c_R(i))$ if $c_R(i) \neq \varepsilon$ and $R(i) := \varepsilon$ otherwise. The subtrees rooted at the left and right child of the root are denoted by $L(T)$ and $R(T)$, respectively, i.e., we have $L(T) = L(r(T))$, and similarly $R(T) = R(r(T))$. A *left path* is a binary tree in which no vertex has a right child. A *left branch* in a binary tree is a subtree that is isomorphic to a left path. The notions *right path* and *right branch* are defined analogously, by interchanging left and right.

We associate $T \in \mathcal{T}_n$ with a permutation $\tau(T)$ of $[n]$ defined by

$$\tau(T) := \big(r(T), \tau(L(T)), \tau(R(T))\big), \tag{1}$$

where the base case of the empty tree ε is defined to be the empty permutation $\tau(\varepsilon) := \varepsilon$. In words, $\tau(T)$ is the sequence of vertex labels obtained from a preorder traversal of T, i.e., we first record the label of the root and then recursively record labels of its left subtree followed by labels of its right subtree. Note that the right path $T \in \mathcal{T}_n$ satisfies $\tau(T) = \mathrm{id}_n$, the identity permutation.

For any vertex i we let $\beta_R(i)$ denote the number of vertices on the right branch starting at i, with the special case $\beta_R(\varepsilon) := 0$. We also define $B_R^-(i) := \{c_R^{j-1}(i) \mid j = 1, \ldots, \beta_R(i) - 1\}$ as the corresponding sets of vertices on this branch except the last one.

2.2 Pattern-avoiding binary trees

Our notion of pattern avoidance in binary trees generalizes the two distinct notions considered in [38] and [11] (recall Figure 1). This definition is illustrated in Figure 5. A *tree pattern* is a pair (P, e) where $P \in \mathcal{T}_k$ and $e\colon [k] \setminus r(P) \to \{0, 1\}$. For any vertex $i \in [k] \setminus r(P)$, a value $e(i) = 0$ is interpreted as the edge leading from i to its parent $p(i)$ being non-contiguous, whereas a value $e(i) = 1$ is interpreted as this edge being contiguous. In our figures, edges $(i, p(i))$ in P with $e(i) = 1$ are drawn solid, and edges with $e(i) = 0$ are drawn dotted. Formally, a tree $T \in \mathcal{T}_n$ *contains* the pattern (P, e) if there is an injective mapping $f\colon [k] \to [n]$ satisfying the following conditions:

- **(i)** For every edge $(i, p(i))$ of P with $e(i) = 1$, we have that $f(i)$ is a child of $f(p(i))$ in T. Specifically, if $i = c_L(p(i))$ then $f(i)$ is the left child of $f(p(i))$, i.e., we have $f(i) = c_L(f(p(i)))$, whereas if $i = c_R(p(i))$ then $f(i)$ is the right child of $f(p(i))$, i.e., we have $f(i) = c_R(f(p(i)))$.
- **(ii)** For every edge $(i, p(i))$ of P with $e(i) = 0$, we have that $f(i)$ is a descendant of $f(p(i))$ in T. Specifically, if $i = c_L(p(i))$, then $f(i)$ is a left descendant of $f(p(i))$, i.e., we have $f(i) \in L(f(p(i)))$, whereas if $i = c_R(p(i))$, then $f(i)$ is a right descendant of $f(p(i))$, i.e., we have $f(i) \in R(f(p(i)))$.

We can retrieve the notions of contiguous and non-contiguous pattern containment used in [38] and [11] as special cases by defining $e(i) := 1$ for all $i \in [k] \setminus r(P)$, or $e(i) := 0$ for all $i \in [k] \setminus r(P)$, respectively.

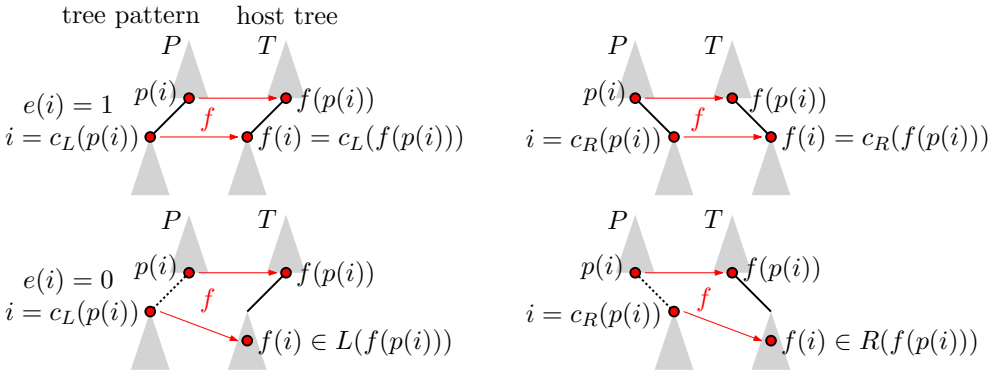


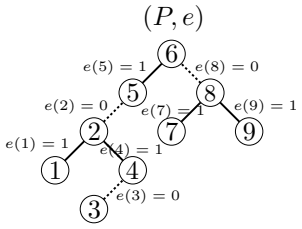
Figure 5 Illustration of our notion of pattern containment in binary trees.

If T does not contain (P, e), then we say that T *avoids* (P, e). Furthermore, we define the set of binary trees with n vertices that avoid the pattern (P, e) as

$$\mathcal{T}_n(P, e) := \{T \in \mathcal{T}_n \mid T \text{ avoids } (P, e)\}.$$

For avoiding multiple patterns $(P_1, e_1), \ldots, (P_\ell, e_\ell)$ simultaneously, we define

$$\mathcal{T}_n\big((P_1, e_1), \ldots, (P_\ell, e_\ell)\big) := \bigcap_{i=1}^{\ell} \mathcal{T}_n(P_i, e_i).$$



$\tau(P) = 652143879$
$e = (e(5), e(2), e(1), e(4), e(3),$
$\quad e(8), e(7), e(9)) = 10110011$
$(P, e) = (652143879, 10110011)$

Figure 6 Compact encoding of binary tree patterns.

We often write a tree pattern (P, e), $P \in \mathcal{T}_k$, in compact form as a pair $\big(\tau(P), (e(\tau_2), \ldots, e(\tau_k))\big)$ where $\tau(P) = (\tau_1, \tau_2, \ldots, \tau_k)$; see Figure 6. In words, the tree P is specified by the preorder permutation $\tau(P)$, and the function e is specified by the sequence of values for all vertices except the root in the preorder sequence, i.e., this sequence has length $k - 1$.

For any tree pattern (P, e), we write $\mu(P, e)$ for the pattern obtained by mirroring the tree, i.e., by changing left and right. Note that the mirroring operation changes the vertex labels so that the search tree property is maintained, specifically the vertex i becomes $n+1-i$. Trivially, we have $\mathcal{T}_n(\mu(P, e)) = \mu(\mathcal{T}_n(P, e))$, in particular (P, e) and $\mu(P, e)$ are Wilf-equivalent.

3 Encoding binary trees by permutations

In this section we establish that avoiding a tree pattern in binary trees is equivalent to avoiding a corresponding mesh pattern in 231-avoiding permutations (Theorem 2 below).

3.1 Pattern-avoiding permutations

We write S_n for the set of all permutations of $[n]$. Given two permutations $\pi \in S_n$ and $\tau \in S_k$, we say that π *contains* τ *as a pattern* if there is a sequence of indices $\nu_1 < \cdots < \nu_k$, such that $\pi(\nu_1), \ldots, \pi(\nu_k)$ are in the same relative order as $\tau = \tau(1), \ldots, \tau(k)$. If π does not contain τ, then we say that π *avoids* τ. We write $S_n(\tau)$ for the permutations from S_n that avoid the pattern τ. More generally, for multiple patterns $\tau_1, \ldots, \tau_\ell$ we define $S_n(\tau_1, \ldots, \tau_\ell) := \bigcap_{i=1}^{\ell} S_n(\tau_i)$, i.e., this is the set of permutations of length n that avoid each of the patterns $\tau_1, \ldots, \tau_\ell$. It is well known that preorder traversals of binary trees are in bijection with 231-avoiding permutations (see, e.g. [27]).

▶ **Lemma 1.** *The mapping* $\tau\colon \mathcal{T}_n \to S_n(231)$ *defined in* (1) *is a bijection.*

3.2 Mesh patterns

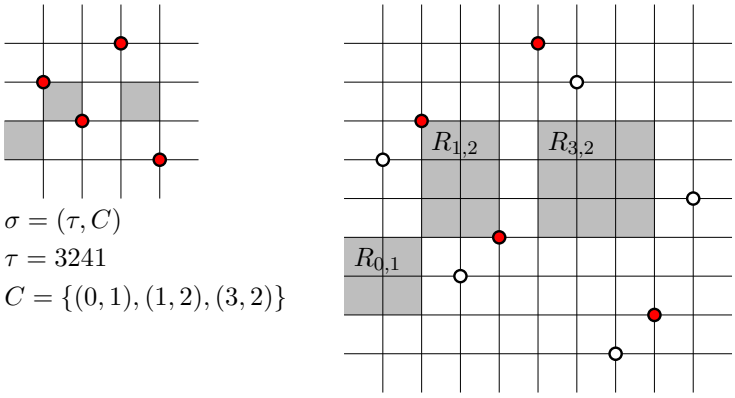


Figure 7 Illustration of mesh pattern containment.

Mesh patterns were introduced by Brändén and Claesson [9], and they generalize classical permutation patterns discussed in the previous section. We recap the required definitions; see Figure 7. The *grid representation* of a permutation $\pi \in S_n$ is defined as $G(\pi) := \{(i, \pi(i)) \mid i \in [n]\}$. Graphically, this is the permutation matrix corresponding to π.

A *mesh pattern* is a pair $\sigma := (\tau, C)$, where $\tau \in S_k$ and $C \subseteq \{0, \ldots, k\} \times \{0, \ldots, k\}$. In our figures, we depict σ by the grid representation of τ, and we shade all unit squares $[i, i+1] \times [j, j+1]$ for which $(i, j) \in C$. A permutation $\pi \in S_n$ *contains* the mesh pattern $\sigma = (\tau, C)$, if there is a sequence of indices $\nu_1 < \cdots < \nu_k$ such that the following two conditions hold:

- **(i)** The entries of $\pi(\nu_1), \ldots, \pi(\nu_k)$ are in the same relative order as $\tau = \tau(1), \ldots, \tau(k)$.
- **(ii)** We let $\lambda_1 < \cdots < \lambda_k$ be the values $\pi(\nu_1), \ldots, \pi(\nu_k)$ sorted in increasing order. For all pairs $(i, j) \in C$, we require that $G(\pi) \cap R_{i,j} = \emptyset$, where $R_{i,j}$ is the rectangular open set defined as $R_{i,j} := (\nu_i, \nu_{i+1}) \times (\lambda_j, \lambda_{j+1})$, using the sentinel values $\nu_0 := \lambda_0 := 0$ and $\nu_{k+1} = \lambda_{k+1} := n + 1$.

The first condition requires a match of the classical pattern τ in π. The second condition requires that $G(\pi)$ has no point in any of the regions $R_{i,j}$ that correspond to the shaded cells C of the pattern. Thus, the classical pattern $\tau \in S_k$ is the mesh pattern $(\tau, \emptyset)$.

3.3 From binary tree patterns to mesh patterns

In the following, for a given tree pattern (P, e), $P \in \mathcal{T}_k$, we construct a permutation mesh pattern $\sigma(P, e) = (\tau(P), C)$, consisting of the permutation $\tau(P)$ obtained by a preorder traversal of the tree P and a set of shaded cells C. These definitions are illustrated in Figures 8 and 9. We consider the inverse permutation of $\tau(P) \in S_k$, which we abbreviate to $\rho := \tau(P)^{-1} \in S_k$. The permutation ρ gives the position of each vertex in the preorder traversal $\tau(P)$ of P. Recall the definition of the set $B_R^-(i)$ given in Section 2.1. For any vertex $i \in [k]$ we define

$$C_i := \big\{(\rho(i) - 1, j) \mid j \in B_R^-(i)\big\}, \tag{2a}$$

and for any $i \in [k] \setminus r(P)$ we define

$$C'_i := \begin{cases} \emptyset & \text{if } e(i) = 0, \\ \big\{\big(\rho(i) - 1, \min P(i) - 1\big),\ \big(\rho(i) - 1, \max P(i)\big)\big\} & \text{if } e(i) = 1. \end{cases} \tag{2b}$$

Then the mesh pattern $\sigma(P, e)$ corresponding to the tree pattern (P, e) is defined as

$$\sigma(P, e) := \Big(\tau(P), \bigcup\nolimits_{i \in [k]} C_i \cup \bigcup\nolimits_{i \in [k] \setminus r(P)} C'_i\Big). \tag{2c}$$

In words, for every pair of vertices (not necessarily distinct and not necessarily forming an edge) except the last vertex on a maximal right branch we shade the cell directly left of the smaller vertex and directly above the larger vertex, and for every edge $(i, p(i))$ with $e(i) = 1$ we shade two additional cells to the left and bottom/top of the submatrix corresponding to the subtree $P(i)$.

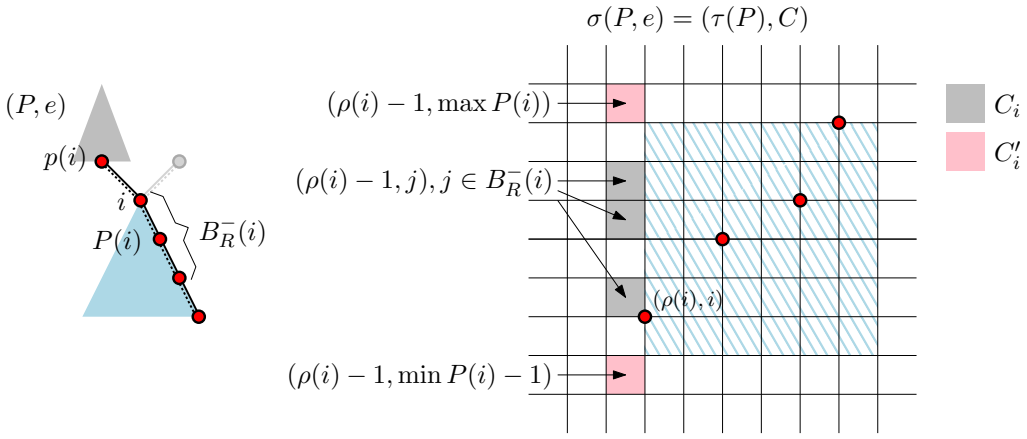


Figure 8 Schematic illustration of the definition of the mesh pattern $\sigma(P, e)$ for a tree pattern (P, e). The edges of the tree P can be contiguous or non-contiguous, and are therefore drawn half solid and half dotted. In the tree shown in the figure, i is the right child of $p(i)$, but it might also be the left child of $p(i)$ (faint lines). On the right, the shaded cells belong to the mesh pattern, and the hatched region corresponds to the submatrix given by the subtree $P(i)$.

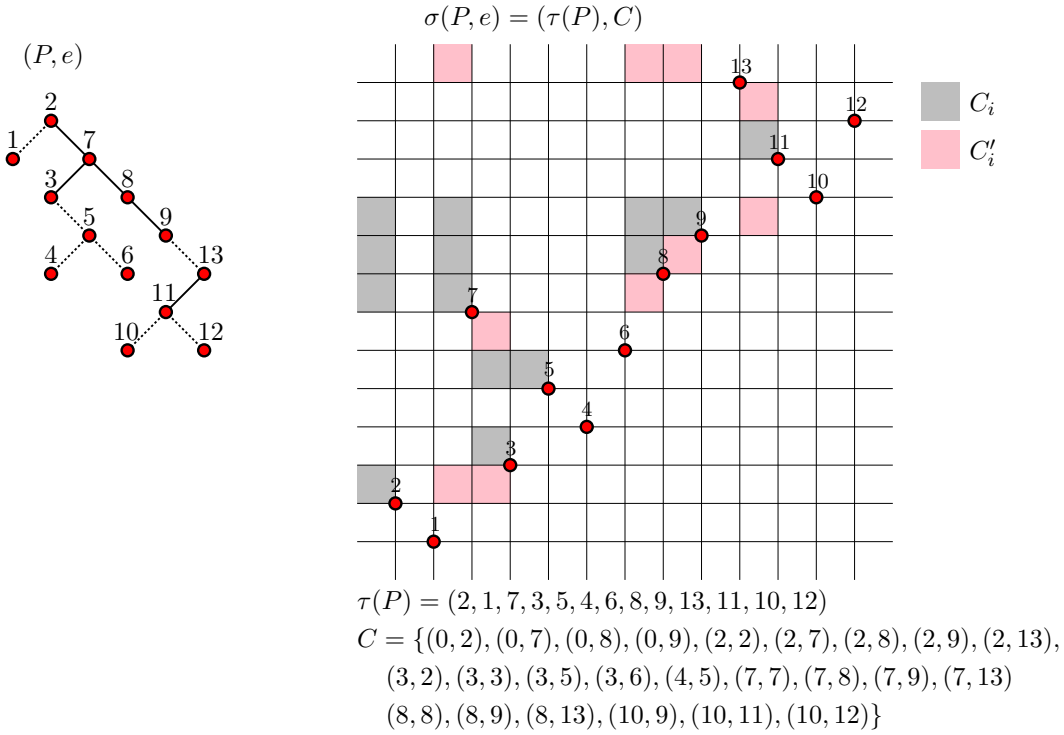


Figure 9 Specific example of the mesh pattern $\sigma(P, e)$ corresponding to a tree pattern (P, e).

The following generalization of Lemma 1 is the main result of this section. Our theorem also generalizes Theorem 12 from [37], which is obtained as the special case when all edges of P are non-contiguous, i.e., $e(i) = 0$ for all $i \in [k] \setminus r(P)$.

▶ **Theorem 2.** *For any tree pattern (P, e), $P \in \mathcal{T}_k$, consider the mesh pattern $\sigma(P, e) = (\tau(P), C)$ defined in (2). Then the mapping $\tau \colon \mathcal{T}_n(P, e) \to S_n(231, \sigma(P, e))$ is a bijection.*

This theorem extends naturally to avoiding multiple tree patterns $(P_1, e_1), \ldots, (P_\ell, e_\ell)$, i.e., $\tau \colon \mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell)) \to S_n(231, \sigma(P_1, e_1), \ldots, \sigma(P_\ell, e_\ell))$ is a bijection. The proof of Theorem 2 can be found in [20].

4 Generating pattern-avoiding binary trees

In this section we apply the Hartung–Hoang–Mütze–Williams generation framework to pattern-avoiding binary trees. The main results are simple and efficient algorithms (Algorithm S and Algorithm H) to generate different classes of pattern-avoiding binary trees, subject to some mild constraints on the tree pattern(s) that are inherited from applying the framework (Theorems 3 and 4, respectively).

4.1 Tree rotations and slides

A natural and well-studied operation on binary trees are tree rotations; see Figure 2. We consider a tree $T \in \mathcal{T}_n$ and one of its edges (i, j) with $j = c_R(i)$, and we let Y be the left subtree of j, i.e., $Y := L(j)$. A *rotation of the edge (i, j)* yields the tree obtained by the following modifications: The child i of $p(i)$ is replaced by j (unless $p(i) = \varepsilon$ in T), i becomes

the left child of j, and Y becomes the right subtree of i. We denote this operation by $j\triangle$, and we refer to it as *up-rotation of j*, indicating that the vertex j moves up. The operation $j\triangle$ is well-defined if and only if j is not the root and $p(j) < j$, or equivalently $j = c_R(p(j))$. The inverse operation is denoted by $j\triangledown$, and we refer to it as *down-rotation of j*, indicating that the vertex j moves down. The operation $j\triangledown$ is well-defined if and only if j has a left child (which must be smaller), i.e., $c_L(j) \neq \varepsilon$. An *up-slide* or *down-slide of j by d steps* is a sequence of d up- or down-rotations of j, respectively, which we write as $(j\triangle)^d$ and $(j\triangledown)^d$.

4.2 A simple greedy algorithm

We use the following simple greedy algorithm to generate a set of binary trees $\mathcal{L}_n \subseteq \mathcal{T}_n$. We say that a slide is *minimal* (w.r.t. $\mathcal{L}_n$), if every slide of the same vertex in the same direction by fewer steps creates a binary tree that is not in $\mathcal{L}_n$.

Algorithm S *(Greedy slides).* This algorithm attempts to greedily generate a set of binary trees $\mathcal{L}_n \subseteq \mathcal{T}_n$ using minimal slides starting from an initial binary tree $T_0 \in \mathcal{L}_n$.

S1. [Initialize] Visit the initial tree T_0.

S2. [Slide] Generate an unvisited binary tree from $\mathcal{L}_n$ by performing a minimal slide of the largest possible vertex in the most recently visited binary tree. If no such slide exists, or the direction of the slide is ambiguous, then terminate. Otherwise visit this binary tree and repeat S2.

To illustrate the algorithm, consider the example in Figure 10. Suppose we choose the right path T_1 shown in the figure as initial tree for the algorithm, i.e., $T_0 := T_1$. In the first iteration, Algorithm S performs an up-slide of the vertex 4 by three steps to obtain T_2. This up-slide is minimal, as an up-slide of 4 in T_1 by one or two steps creates the forbidden tree pattern (P, e). Note that any tree created from T_2 by a down-slide of 4 either contains the forbidden pattern or has been visited before. Consequently, the algorithm applies an up-slide of 3 by two steps, yielding T_3. After five more slides, the algorithm terminates with T_8, and at this point it has visited all eight trees in $\mathcal{T}_4(P, e)$.

Now consider the example in Figure 11, where the algorithm terminates after having visited six different trees from $\mathcal{T}_4(P, e)$. However, the set $\mathcal{T}_4(P, e)$ contains two more trees that are not visited by the algorithm.

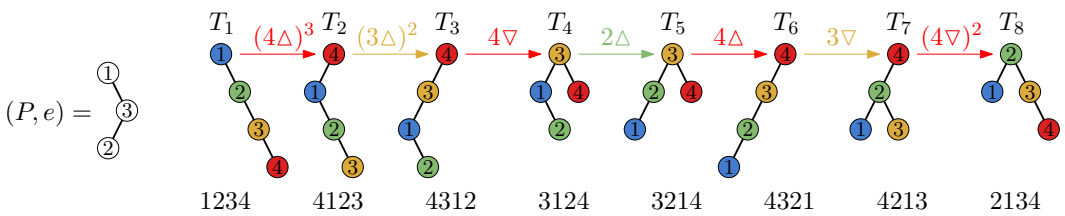


Figure 10 Run of Algorithm S that visits all binary trees in the set $\mathcal{T}_4(P, e)$. Below each tree T is the corresponding permutation $\tau(T)$.

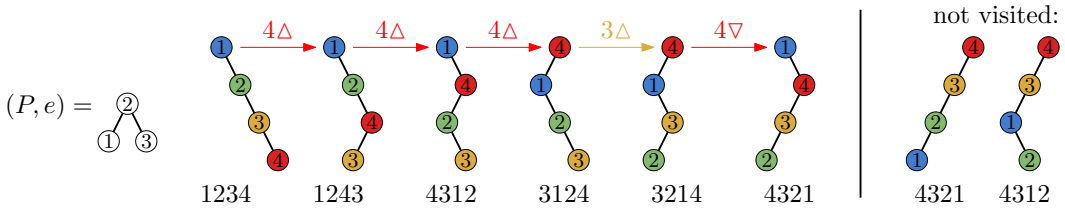


Figure 11 Run of Algorithm S that does not visit all binary trees in the set $\mathcal{T}_4(P, e)$.

We now formulate simple sufficient conditions on the tree pattern (P, e) ensuring that Algorithm S successfully visits all trees in $\mathcal{T}_n(P, e)$. Specifically, we say that a tree pattern (P, e), $P \in \mathcal{T}_k$, is *friendly*, if it satisfies the following three conditions; see Figure 12:

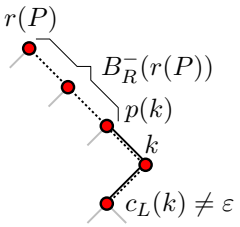


Figure 12 Definition of friendly tree patterns.

(i) We have $p(k) \neq \varepsilon$ and $c_L(k) \neq \varepsilon$, i.e., the largest vertex k is neither the root nor a leaf in P.

(ii) For every $j \in B_R^-(r(P)) \setminus r(P)$ we have $e(j) = 0$, i.e., the edges on the right branch starting at the root, except possibly the last one, are all non-contiguous.

(iii) If $e(k) = 1$, then we have $e(c_L(k)) = 0$, i.e., if the edge from k to its parent is contiguous, then the edge to its left child must be non-contiguous.

Note that for non-contiguous tree patterns, i.e., $e(i) = 0$ for all $i \in [k] \setminus r(P)$, conditions (ii) and (iii) are always satisfied. The following is our main result of this section.

▶ **Theorem 3.** *Let $(P_1, e_1), \ldots, (P_\ell, e_\ell)$ be friendly tree patterns. Then Algorithm S initialized with the tree $\tau^{-1}(\mathrm{id}_n)$ visits every binary tree from $\mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell))$ exactly once.*

Recall that $\tau^{-1}(\mathrm{id}_n)$ is the right path, i.e., the tree that corresponds to the identity permutation. Note that by condition (i) in the definition of friendly tree pattern, we have $\tau^{-1}(\mathrm{id}_n) \in \mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell))$. Theorem 3 can be proved by applying the Hartung–Hoang–Mütze–Williams generation framework [21]; see [20] for details. In particular, our notion of friendly tree patterns is inherited from the notion of tame mesh permutation patterns used in [21, Thm. 15].

4.3 Efficient implementation

We now describe an efficient implementation of Algorithm S. In particular, this implementation is *history-free*, i.e., it does not require to store all previously visited binary trees, but only maintains the current tree in memory. Algorithm H is a straightforward translation of the history-free Algorithm M presented in [34] from permutations to binary trees.

Algorithm H *(History-free minimal slides).* For friendly tree patterns $(P_1, e_1), \ldots, (P_\ell, e_\ell)$, this algorithm generates all binary trees from $\mathcal{T}_n$ that avoid $(P_1, e_1), \ldots, (P_\ell, e_\ell)$, i.e., the set $\mathcal{L}_n := \mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell)) \subseteq \mathcal{T}_n$ by minimal slides in the same order as Algorithm S. It maintains the current tree in the variable T, and auxiliary arrays $o = (o_1, \ldots, o_n)$ and $s = (s_1, \ldots, s_n)$.

H1. [Initialize] Set $T \leftarrow \tau^{-1}(\mathrm{id}_n)$, and $o_j \leftarrow \triangle$, $s_j \leftarrow j$ for $j = 1, \ldots, n$.

H2. [Visit] Visit the current binary tree T.

H3. [Select vertex] Set $j \leftarrow s_n$, and terminate if $j = 1$.

H4. [Slide] In the current binary tree T, perform a slide of the vertex j that is minimal w.r.t. $\mathcal{L}_n$, where the slide direction is up if $o_j = \triangle$ and down if $o_j = \triangledown$.

H5. [Update o and s] Set $s_n \leftarrow n$. If $o_j = \triangle$ and j is either the root or its parent is larger than j set $o_j = \triangledown$, or if $o_j = \triangledown$ and j has no left child set $o_j = \triangle$, and in both cases set $s_j \leftarrow s_{j-1}$ and $s_{j-1} = j - 1$. Go back to H2.

The two auxiliary arrays used by Algorithm H store the following information. The direction in which vertex j slides in the next step is maintained in the variable o_j. Furthermore, the array s is used to determine the vertex that slides in the next step. Specifically, the vertex j that slides in the next steps is retrieved from the last entry of the array s in step H3, by the instruction $j \leftarrow s_n$. The running time per iteration of the algorithm is governed by the time it takes to compute a minimal slide in step H4. This boils down to testing containment of the tree patterns (P_i, e_i), $i \in [\ell]$, in T.

▶ **Theorem 4.** *Let $(P_1, e_1), \ldots, (P_\ell, e_\ell)$ be friendly tree patterns with $P_i \in \mathcal{T}_{k_i}$ for $i \in [\ell]$. Then Algorithm H visits every binary tree from $\mathcal{T}_n((P_1, e_1), \ldots, (P_\ell, e_\ell))$ exactly once, in the same order as Algorithm S, in time $\mathcal{O}(n^2 \sum_{i=1}^{\ell} k_i^2)$ per binary tree.*

See [20] for a proof of Theorem 4.

5 Tree patterns on at most 5 vertices

We conducted systematic computer experiments with all tree patterns (P, e) on $k = 3, 4, 5$ vertices; see Tables 1, 2 and 3, respectively. Specifically, we computed the corresponding counting sequences $|\mathcal{T}_n(P, e)|$ for $n = 1, \ldots, 12$, and searched for matches within the OEIS [36]. There are three new counting sequences denoted by `NewA`, `NewB`, and `NewC`, which we added to the OEIS using the sequence numbers A365508, A365509, and A365510, respectively. All those counts were computed using Algorithm H for friendly tree patterns, and via brute-force methods for non-friendly tree patterns. As mirrored tree patterns are Wilf-equivalent, our tables only contain the lexicographically smaller of any such pair of mirrored trees, using the compact encoding described in Section 2.2.

It turns out that for some edges $(i, p(i))$ in a tree pattern (P, e), it is irrelevant whether the edge is considered contiguous ($e(i) = 1$) or non-contiguous ($e(i) = 0$). We have a theorem ([20, Thm. 11]) that describes these situations, and this theorem is used heavily in our tables, where those "don't care" values of e are denoted by the hyphen `-`. The statement and proof of this theorem are slightly technical, and so we omit it in this extended abstract.

Table 1 Tree patterns with 3 vertices. See Section 5 for explanations.

P	e	Friendly	Counts $\lvert\mathcal{T}_n(P,e)\rvert$ for $n=1,\ldots,12$													OEIS
123	0-		1	2	4	8	16	32	64	128	256	512	1024	2048	...	A000079
	1-		1	2	4	9	21	51	127	323	835	2188	5798	15511	...	A001006
132	--	0-, -0	1	2	4	8	16	32	64	128	256	512	1024	2048	...	A000079
213	--		1	2	4	8	16	32	64	128	256	512	1024	2048	...	A000079

Table 2 Tree patterns with 4 vertices.

P	e	Friendly	Counts $\lvert\mathcal{T}_n(P,e)\rvert$ for $n=1,\ldots,12$													OEIS
1234	00-		1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	01-		1	2	5	13	35	96	267	750	2123	6046	17303	49721	...	A005773
	10-		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
	11-		1	2	5	13	36	104	309	939	2905	9118	28964	92940	...	A036765
1243	0--	00-, 0-0	1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	1--		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
1324	0--		1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	1--		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
1423	0--, -0-	0--, -0-	1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	11-		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
1432	-0-	-0-	1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	-1-	01-	1	2	5	13	35	96	267	750	2123	6046	17303	49721	...	A005773
2134	-0-		1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	-1-		1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242
2143	-0-	-0-	1	2	5	13	34	89	233	610	1597	4181	10946	28657	...	A001519
	-1-	-10	1	2	5	13	35	97	275	794	2327	6905	20705	62642	...	A025242

Table 3 Tree patterns with 5 vertices.

P	e	Friendly	Counts $\lvert\mathcal{T}_n(P,e)\rvert$ for $n=1,\ldots,12$													OEIS
12345	000-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	001-		1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	010-		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	011-		1	2	5	14	41	124	384	1210	3865	12482	40677	133572	...	A159772
	100-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	101-		1	2	5	14	41	124	383	1202	3819	12255	39651	129190	...	NewB→A365509
	110-		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768
	111-		1	2	5	14	41	125	393	1265	4147	13798	46476	158170	...	A036766
12354	00--	000-, 00-0	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	11--		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768
12435	00--		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	11--		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768
12534	00--, 0-0-	00--, 0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	011-		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--, 1-0-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	111-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
12543	0-0-	0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-	001-	1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	1-0-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	101-		1	2	5	14	41	124	383	1202	3819	12255	39651	129190	...	NewB→A365509
	111-		1	2	5	14	41	124	384	1211	3875	12548	41040	135370	...	A159770
13245	0-0-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	1-0-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	1-1-		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768

P	e	Friendly	Counts $\lvert\mathcal{T}_n(P,e)\rvert$ for $n=1,\ldots,12$													OEIS
13254	0-0-	0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-	0-10	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	1-0-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	1-1-		1	2	5	14	41	124	385	1220	3929	12822	42309	140922	...	A159771
14235	00--		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	11--		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
14325	00--		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--		1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	10--		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	11--		1	2	5	14	41	124	384	1211	3875	12548	41040	135370	...	A159770
15234	0-0-, -00-	0-0-, -00-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-, -01-	0-1-, -01-	1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	110-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	111-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
15243	-0--, 0-0-	-0--, 0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	011-	011-	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	110-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	111-		1	2	5	14	41	124	385	1220	3929	12822	42309	140922	...	A159771
15324	-0--	-0--	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--	01--	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	11--		1	2	5	14	41	124	384	1211	3875	12548	41040	135370	...	A159770
15423	-0--	-0--	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	01--	01--	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	-10-	010-	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	111-		1	2	5	14	41	124	384	1211	3875	12548	41040	135370	...	A159770
15432	-00-	-00-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-01-	-01-	1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	-10-	010-	1	2	5	14	41	123	375	1157	3603	11304	35683	113219	...	NewA→A365508
	-11-	011-	1	2	5	14	41	124	384	1210	3865	12482	40677	133572	...	A159772
21345	-00-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-01-		1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	-10-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	-11-		1	2	5	14	41	124	385	1221	3939	12886	42648	142544	...	A159768
21354	-0--	-00-, -0-0	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-10-		1	2	5	14	41	123	376	1168	3678	11716	37688	122261	...	NewC→A365510
	-11-		1	2	5	14	41	124	384	1212	3885	12613	41389	137055	...	A159773
21435	-0--		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-1--		1	2	5	14	41	124	385	1220	3929	12822	42309	140922	...	A159771
21534	-0--	-0--	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-10-	-10-	1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	-11-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
21543	-00-	-00-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	-01-	-01-	1	2	5	14	41	123	374	1147	3538	10958	34042	105997	...	A054391
	-10-	-10-	1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	-11-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
31245	0-0-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	1-0-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	1-1-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
31254	0-0-	0-0-	1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-, 1-0-	0-10, 1-0-	1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	1-1-	1-10	1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769
32145	0-0-		1	2	5	14	41	122	365	1094	3281	9842	29525	88574	...	A007051
	0-1-, 1-0-		1	2	5	14	41	123	375	1158	3615	11393	36209	115940	...	A176677
	1-1-		1	2	5	14	41	124	384	1212	3885	12614	41400	137132	...	A159769

6 Bijections between binary trees and Motzkin paths

In this section, we present bijections between pattern-avoiding binary trees and different types of Motzkin paths. For more bijections with other combinatorial objects, see [20]. Specifically, we consider lattice paths with steps $\mathtt{U} := (1,1)$, $\mathtt{D} := (1,-1)$, $\mathtt{F} := (1,0)$, and $\mathtt{D}_h := (1,-h)$ for $h \geq 2$. An *n-step Motzkin path* starts at $(0,0)$, ends at $(n,0)$, uses only steps `U`, `D` or `F`, and it never goes below the x-axis. We write $\mathcal{M}_n$ for the set of all n-step Motzkin paths (OEIS A001006). An *n-step Motzkin left factor* starts at $(0,0)$, uses n many steps `U`, `D` or `F`,

and it never goes below the x-axis. We write $\mathcal{L}_n$ for the set of all n-step Motzkin left factors (OEIS A005773). An *n-step Motzkin path with catastrophes* [4] starts at $(0,0)$, ends at $(n,0)$, uses only steps U, D, F, or $\mathtt{D}_h$ for $h \geq 2$, such that all $\mathtt{D}_h$-steps end on the x-axis, and it never goes below the x-axis (OEIS A054391). We write $\mathcal{C}_n$ for the set of all n-step Motzkin paths with catastrophes.

6.1 Bijection between $\mathcal{T}_n(123, 1\text{-})$ and Motzkin paths $\mathcal{M}_n$

This bijection is illustrated in Figure 13 (a). Consider a tree $T \in \mathcal{T}_n(P,e)$ where $(P,e) := (123, 1\text{-})$. Due to the forbidden pattern (P,e), every maximal right branch in T consists of one or two vertices, but not more. We map T to an n-step Motzkin path $f(T)$ as follows. Every maximal right branch in T consisting of one vertex i creates an F-step at position i in $f(T)$. Every maximal right branch in T consisting of two vertices i and j, where $j = c_R(i)$, creates a pair of U-step and D-step at the same height at positions i and j in $f(T)$, respectively. It is easy to verify that f is indeed a bijection between $\mathcal{T}_n(P,e)$ and $\mathcal{M}_n$.

Rowland [38] described a bijection between $\mathcal{T}_n(123, 1\text{-})$ and $\mathcal{M}_n$ that is different from f.

6.2 Bijection between $\mathcal{T}_n(1432, \text{-}1\text{-})$ and Motzkin left factors $\mathcal{L}_{n-1}$

This bijection is illustrated in Figure 13 (b), and it uses as a building block the bijection f defined in the previous section. Instead of $(1432, \text{-}1\text{-})$, we consider the mirrored tree pattern $(P,e) := \mu(1432, \text{-}1\text{-}) = (4123, \text{-}1\text{-})$ for convenience. Consider a tree $T \in \mathcal{T}_n(P,e)$. We define $b := \beta_R(r(T))$ and $r_i := c_R^{i-1}(r(T))$ for $i = 1, \ldots, b$, i.e., we consider the right branch $(r_1, \ldots, r_b)$ starting from the root of T. Due to the forbidden tree pattern (P,e), each subtree $L(r_i)$ for $i = 1, \ldots, b$ is $(123, 1\text{-})$-avoiding. Using the bijection f described in the previous section, we can thus map each subtree $L(r_i)$ to a Motzkin path $f(L(r_i))$. Therefore, we map T to an $(n-1)$-step Motzkin left factor $g(T)$ by combining the subpaths $f(L(r_i))$, separating them by in total $b-1$ many U-steps, one between every two consecutive subpaths $f(L(r_i))$ and $f(L(r_{i+1}))$. To make the proof work, the subpaths $f(L(r_i))$ can be combined in increasing order from left to right on $g(T)$, i.e., for $i = 1, \ldots, b$, or in decreasing order, i.e., for $i = b, b-1, \ldots, 1$, and for reasons that will become clear in the next section we combine them in decreasing order, i.e.,

$$g(T) := f(L(r_b)), \mathtt{U}, f(L(r_{b-1})), \ldots, \mathtt{U}, f(L(r_1)). \tag{3}$$

The mapping g is clearly a bijection between $\mathcal{T}_n(P,e)$ and $\mathcal{L}_{n-1}$.

6.3 Bijection between $\mathcal{T}_n(21543, \text{-}01\text{-})$ and Motzkin paths with catastrophes $\mathcal{C}_n$

This bijection is illustrated in Figure 13 (c), and it uses as a building block the bijection g defined in the previous section. Instead of $(21543, \text{-}01\text{-})$, we consider the mirrored tree pattern $(P,e) := \mu(21543, \text{-}01\text{-}) = (41235, 01\text{--})$ for convenience. Consider a tree $T \in \mathcal{T}_n(P,e)$ and the rightmost leaf in T, and partition the path from the root of T to that leaf into a sequence of maximal right branches $B_1, \ldots, B_\ell$. For $i = 1, \ldots, \ell$, we let T_i be the subtree of T that consists of B_i plus the left subtrees of all vertices on B_i except the last one. Note that $T_1, \ldots, T_\ell$ form a partition of T. Furthermore, T avoiding (P,e) is equivalent to each of the T_i, $i = 1, \ldots, \ell$, avoiding $(4123, 01\text{-})$. Using the bijection g described in the previous section, we can thus map each subtree T_i to a Motzkin left factor $g(T_i)$, and by appending one additional appropriate step F, D or $\mathtt{D}_h$ for $h \geq 2$ we obtain a Motzkin path $g'(T_i)$. Note that

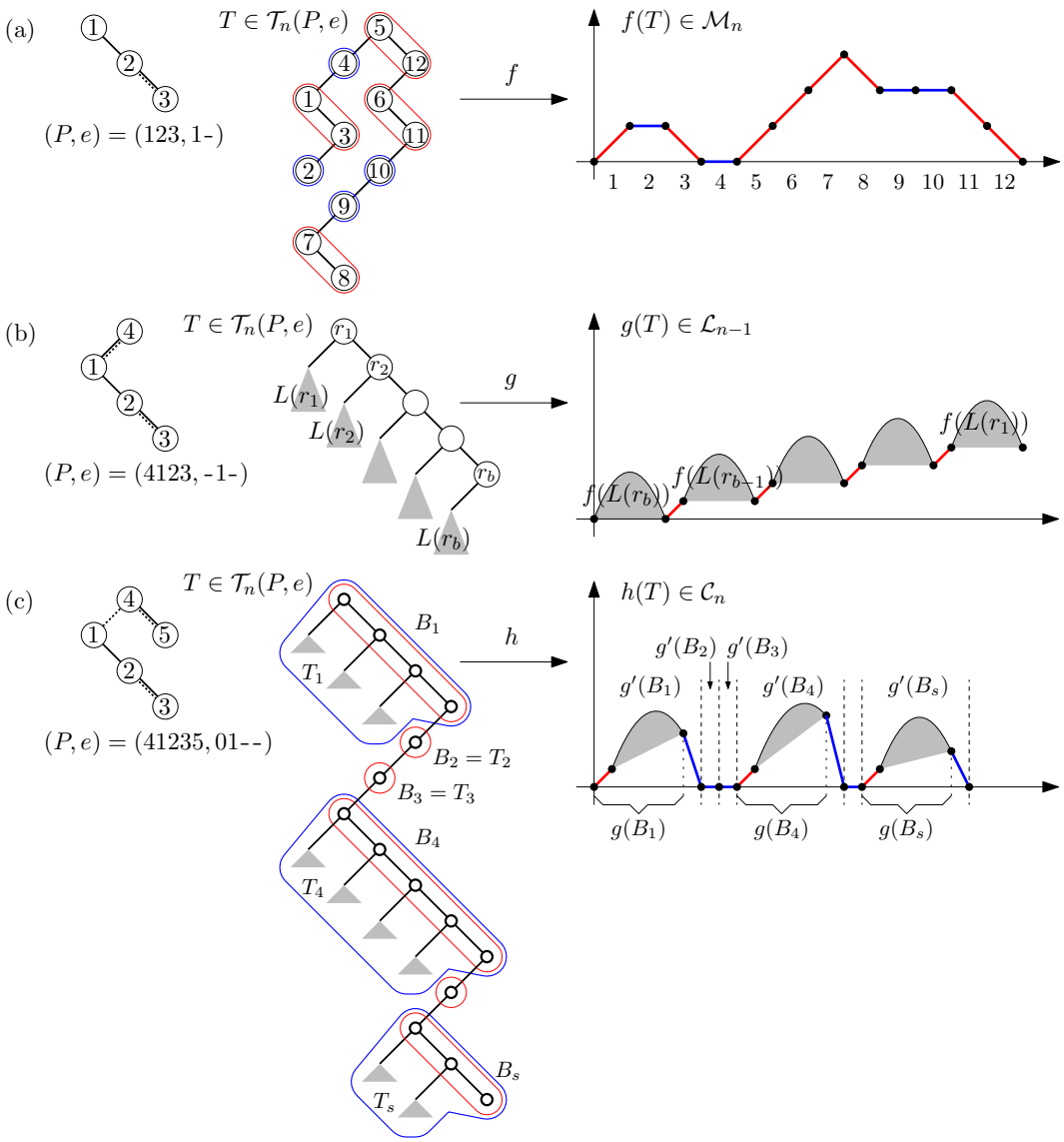


Figure 13 Bijections between pattern-avoiding binary trees and different types of Motzkin paths. Edges $(i, p(i))$ in the tree patterns that can be contiguous or non-contiguous (giving the same pattern-avoiding trees) are drawn as a double line that is half solid and half dotted.

the rightmost leaf of T_i has no left child, and thus the definition (3) yields that $g'(T_i)$ touches the x-axis only at the first point and last point, but at no intermediate (integer) points. Therefore, we map T to an n-step Motzkin path with catastrophes $h(T)$ by concatenating the Motzkin subpaths $g'(T_i)$ for $i = 1, \ldots, \ell$, i.e., $h(T) := g'(T_1), g'(T_2), \ldots, g'(T_\ell)$. It can be readily checked that h is a bijection between $\mathcal{T}_n(P, e)$ and $\mathcal{C}_n$.

7 Open Problems

- Are there elegant bijections between pattern-avoiding binary trees and other interesting combinatorial objects such as Motzkin paths with 2-colored F-steps at odd heights (OEIS A176677), or so-called skew Motzkin paths (OEIS A025242)?

- For purely contiguous or non-contiguous tree patterns (P, e), there are recursions to derive the generating function for $|\mathcal{T}_n(P, e)|$; see [38] and [11]. For our more general mixed tree patterns, these methods seem to fail. Is there is an algorithm to compute those more general generating functions, and what are their properties? Furthermore, can the set of pattern-avoiding trees for such pure (non-friendly) patterns be generated efficiently?
- In addition to contiguous and non-contiguous edges $(i, p(i))$ of a binary tree pattern, which we encode by $e(i) = 1$ and $e(i) = 0$, there is another very natural notion of pattern containment that is intermediate between those two, which we may encode by setting $e(i) := 1/2$. Specifically, for such an edge with $e(i) = 1/2$ in the pattern tree P, we require from the injection f described in Section 2.2 that $f(i)$ is a descendant of $f(p(i))$ along a left or right branch in the host tree T. Specifically, if $i = c_L(p(i))$, then $f(i) = c_L^j(f(p(i)))$ for some $j > 0$, whereas if $i = c_R(p(i))$, then $f(i) = c_R^j(f(p(i)))$ for some $j > 0$. Theorem 2 can be generalized to also capture this new notion, by modifying the definition (2b) in the natural way to

$$C'_i := \begin{cases} \emptyset & \text{if } e(i) = 0, \\ \{(\rho(i) - 1, \min P(i) - 1)\} & \text{if } e(i) = 1/2 \text{ and } i = c_L(p(i)), \\ \{(\rho(i) - 1, \max P(i))\} & \text{if } e(i) = 1/2 \text{ and } i = c_R(p(i)), \\ \{(\rho(i) - 1, \min P(i) - 1), (\rho(i) - 1, \max P(i))\} & \text{if } e(i) = 1. \end{cases}$$

  The notion of friendly tree pattern can be generalized by modifying condition (iii) in Section 4.2 as follows: (iii') If $e(k) \in \{1, 1/2\}$, then we have $e(c_L(k)) \in \{0, 1/2\}$. It is worthwhile to investigate this new notion of pattern containment/avoidance and its interplay with the other two notions. Our computer experiments show that there are patterns with edges $e(i) = 1/2$ that give rise to counting sequences that are distinct from the ones obtained from patterns with edges $e(i) = 1$ (contiguous) and $e(i) = 0$ (non-contiguous). The corresponding functionality has already been built into our generation tool [10].
  This intermediate notion of pattern-avoidance in binary trees has interesting applications in the context of pattern-avoidance in rectangulations, a line of inquiry that was initiated in [34].

References

1. K. Anders and K. Archer. Rooted forests that avoid sets of permutations. *European J. Combin.*, 77:1–16, 2019. doi:10.1016/j.ejc.2018.10.004.
2. A. Asinowski, A. Bacher, C. Banderier, and B. Gittenberger. Analytic combinatorics of lattice paths with forbidden patterns: enumerative aspects. In *Language and automata theory and applications*, volume 10792 of *Lecture Notes in Comput. Sci.*, pages 195–206. Springer, Cham, 2018. doi:10.1007/978-3-319-77313-1_15.
3. E. Babson and E. Steingrímsson. Generalized permutation patterns and a classification of the Mahonian statistics. *Sém. Lothar. Combin.*, 44:Art. B44b, 18 pp., 2000.
4. J.-L. Baril and S. Kirgizov. Bijections from Dyck and Motzkin meanders with catastrophes to pattern avoiding Dyck paths. *Discrete Math. Lett.*, 7:5–10, 2021. doi:10.47443/dml.2021.0032.
5. A. Bernini, L. Ferrari, R. Pinzani, and J. West. Pattern-avoiding Dyck paths. In *25th International Conference on Formal Power Series and Algebraic Combinatorics (FPSAC 2013)*, Discrete Math. Theor. Comput. Sci. Proc., AS, pages 683–694. Assoc. Discrete Math. Theor. Comput. Sci., Nancy, 2013.

6 D. Bevan, D. Levin, P. Nugent, J. Pantone, L. Pudwell, M. Riehl, and M. L. Tlachac. Pattern avoidance in forests of binary shrubs. *Discrete Math. Theor. Comput. Sci.*, 18(2):Paper No. 8, 22 pp., 2016.

7 J. Bloom and S. Elizalde. Pattern avoidance in matchings and partitions. *Electron. J. Combin.*, 20(2):Paper 5, 38, 2013. doi:10.37236/2976.

8 J. Bloom and D. Saracino. Pattern avoidance for set partitions à la Klazar. *Discrete Math. Theor. Comput. Sci.*, 18(2):Paper No. 9, 22 pp., 2016. doi:10.46298/dmtcs.1327.

9 P. Brändén and A. Claesson. Mesh patterns and the expansion of permutation statistics as sums of permutation patterns. *Electron. J. Combin.*, 18(2):Paper 5, 14 pp., 2011.

10 The Combinatorial Object Server: Generate binary trees. http://www.combos.org/btree.

11 M. Dairyko, L. Pudwell, S. Tyner, and C. Wynn. Non-contiguous pattern avoidance in binary trees. *Electron. J. Combin.*, 19(3):Paper 22, 21 pp., 2012. doi:10.37236/2099.

12 F. Disanto. Unbalanced subtrees in binary rooted ordered and un-ordered trees. *Sém. Lothar. Combin.*, 68:Art. B68b, 14 pp., 2012.

13 V. Dotsenko. Pattern avoidance in labelled trees, 2011. arXiv:1110.0844.

14 S. Elizalde and M. Noy. Consecutive patterns in permutations. *Adv. in Appl. Math.*, 30:110–125, 2003. Formal power series and algebraic combinatorics (Scottsdale, AZ, 2001). doi:10.1016/S0196-8858(02)00527-4.

15 N. Gabriel, K. Peske, L. Pudwell, and S. Tay. Pattern avoidance in ternary trees. *J. Integer Seq.*, 15(1):Article 12.1.5, 20 pp., 2012.

16 S. Giraudo. Tree series and pattern avoidance in syntax trees. *J. Combin. Theory Ser. A*, 176:105285, 37, 2020. doi:10.1016/j.jcta.2020.105285.

17 A. Godbole, A. Goyt, J. Herdan, and L. Pudwell. Pattern avoidance in ordered set partitions. *Ann. Comb.*, 18(3):429–445, 2014. doi:10.1007/s00026-014-0232-y.

18 A. M. Goyt. Avoidance of partitions of a three-element set. *Adv. in Appl. Math.*, 41(1):95–114, 2008. doi:10.1016/j.aam.2006.07.006.

19 A. M. Goyt and L. K. Pudwell. Avoiding colored partitions of two elements in the pattern sense. *J. Integer Seq.*, 15(6):Article 12.6.2, 17 pp., 2012.

20 P. Gregor, T. Mütze, and Namrata. Combinatorial generation via permutation languages. VI. Binary trees, 2023. Full preprint version of the present article available at arXiv:2306.08420.

21 E. Hartung, H. P. Hoang, T. Mütze, and A. Williams. Combinatorial generation via permutation languages. I. Fundamentals. *Trans. Amer. Math. Soc.*, 375(4):2255–2291, 2022. doi:10.1090/tran/8199.

22 V. Jelínek and T. Mansour. On pattern-avoiding partitions. *Electron. J. Combin.*, 15(1):Research paper 39, 52 pp., 2008. URL: http://www.combinatorics.org/Volume_15/Abstracts/v15i1r39.html.

23 V. Jelínek, T. Mansour, and M. Shattuck. On multiple pattern avoiding set partitions. *Adv. in Appl. Math.*, 50(2):292–326, 2013. doi:10.1016/j.aam.2012.09.002.

24 M. Klazar. On *abab*-free and *abba*-free set partitions. *European J. Combin.*, 17(1):53–68, 1996. doi:10.1006/eujc.1996.0005.

25 M. Klazar. Counting pattern-free set partitions. I. A generalization of Stirling numbers of the second kind. *European J. Combin.*, 21(3):367–378, 2000. doi:10.1006/eujc.1999.0353.

26 M. Klazar. Counting pattern-free set partitions. II. Noncrossing and other hypergraphs. *Electron. J. Combin.*, 7:Research Paper 34, 25 pp., 2000. URL: http://www.combinatorics.org/Volume_7/Abstracts/v7i1r34.html.

27 G. D. Knott. A numbering system for binary trees. *Commun. ACM*, 20(2):113–115, 1977. doi:10.1145/359423.359434.

28 D. E. Knuth. *The Art of Computer Programming. Vol. 1: Fundamental algorithms*. Addison-Wesley, Reading, MA, 1997. Third edition.

29 G. Kreweras. Sur les partitions non croisées d'un cycle. *Discrete Math.*, 1(4):333–350, 1972. doi:10.1016/0012-365X(72)90041-6.

30 D. Levin, L. K. Pudwell, M. Riehl, and A. Sandberg. Pattern avoidance in k-ary heaps. *Australas. J. Combin.*, 64:120–139, 2016.

31 J. M. Lucas, D. Roelants van Baronaigien, and F. Ruskey. On rotations and the generation of binary trees. *J. Algorithms*, 15(3):343–366, 1993. doi:10.1006/jagm.1993.1045.

32 T. Mansour and M. Shattuck. Pattern avoiding partitions and Motzkin left factors. *Cent. Eur. J. Math.*, 9(5):1121–1134, 2011. doi:10.2478/s11533-011-0057-4.

33 T. Mansour and M. Shattuck. Pattern avoiding partitions, sequence A054391 and the kernel method. *Appl. Appl. Math.*, 6(12):397–411, 2011.

34 A. Merino and T. Mütze. Combinatorial generation via permutation languages. III. Rectangulations. *Discrete Comput. Geom.*, 70:51–122, 2023. doi:10.1007/s00454-022-00393-w.

35 T. Mütze. Combinatorial Gray codes – an updated survey. *Electron. J. Combin.*, DS26:93, 2023. doi:10.37236/11023.

36 OEIS Foundation Inc. The on-line encyclopedia of integer sequences, 2023. URL: http://oeis.org.

37 L. Pudwell, C. Scholten, T. Schrock, and A. Serrato. Noncontiguous pattern containment in binary trees. *International Scholarly Research Notices*, 2014, 2014. doi:10.1155/2014/316535.

38 E. S. Rowland. Pattern avoidance in binary trees. *J. Combin. Theory Ser. A*, 117(6):741–758, 2010. doi:10.1016/j.jcta.2010.03.004.

39 B. E. Sagan. Pattern avoidance in set partitions. *Ars Combin.*, 94:79–96, 2010.

40 A. Sapounakis, I. Tasoulas, and P. Tsikouras. Counting strings in Dyck paths. *Discrete Math.*, 307(23):2909–2924, 2007. doi:10.1016/j.disc.2007.03.005.

41 C. Savage. A survey of combinatorial Gray codes. *SIAM Rev.*, 39(4):605–629, 1997. doi:10.1137/S0036144595295272.

42 A. Williams. The greedy Gray code algorithm. In *Algorithms and data structures*, volume 8037 of *Lecture Notes in Comput. Sci.*, pages 525–536. Springer, Heidelberg, 2013. doi:10.1007/978-3-642-40104-6_46.