COMBINATORIAL GENERATION VIA PERMUTATION LANGUAGES.
VI. BINARY TREES

PETR GREGOR, TORSTEN MÜTZE, AND NAMRATA

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.

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 \cite{Knu97} 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 \cite{Row10, Dis12, DPTW12, GPPT12, PSSS14, BLN16, AA19, Gir20}, set partitions \cite{Kre72, Kla96, Kla00a, Kla00b, Goy08, JM08, MS11a, MS11b, Sag10, GP12, JMS13, GGHP14, BS16}, lattice paths \cite{STT07, BFPW13, ABBG18, BK21}, heaps \cite{LPRS16}, matchings \cite{BE13}, and rectangulations \cite{MM23}. In this work, we focus on binary trees, a class of objects that is fundamental within computer science, and also a classical Catalan family.

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 \cite{Row10} considered a \textit{contiguous} notion of pattern containment, where $T$ contains $P$ if $P$ is present as an induced subtree of $T$; see Figure\ref{fig:contiguous}(a). He devised an algorithm to compute the generating function for the number of $n$-vertex binary trees

---

(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, United Kingdom

E-mail addresses: gregor@ktiml.mff.cuni.cz, torsten.mutze@warwick.ac.uk, namrata@warwick.ac.uk.

This work was supported by Czech Science Foundation grant GA 22-15272S.
that avoid $P$, and he showed that this generating function is always algebraic. Dairyko, Pudwell, Tyner, and Wynn [DPTW12] 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 [Row10] and [DPTW12] are the tree analogues of consecutive [EN03] and classical permutation patterns, respectively. Our new notion of mixed patterns is the tree analogue of vincular permutation patterns [BS00], 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

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{rotation}
\caption{Rotation in binary trees.}
\end{figure}
Ruskey \cite{LRvBR93}, 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. The algorithm is an instance of a combinatorial Gray code \cite{Sav97,Müt22}, 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 \( O(1) \) per generated 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. Williams \cite{Wil13} 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.

1.2. **Our results.** It is well known that binary trees are in bijection with 231-avoiding permutations. Our first contribution is to generalize this bijection, by establishing a one-to-one correspondence between mixed binary tree patterns and mesh permutation patterns, a generalization of classical permutation patterns introduced by Brändén and Claesson \cite{BC11}. 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 \cite{HHMW22}, 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 10, 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 \cite{cos}. 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 \cite{oeis} 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 7.2) and set partitions (Theorem 15).

This paper is the sixth installment in a series of papers on generating a large variety of combinatorial objects by encoding them in a unified way via permutations. This algorithmic framework was developed in [HHMW22], and so far has been applied to generate pattern-avoiding permutations [HHMW22], Table 1, lattice congruences of the weak order on permutations [HM21], pattern-avoiding rectangulations [MM23], elimination trees of graphs [CMM22], and acyclic orientations of graphs [CHM+22]. The present paper thus further extends the reach of this framework to pattern-avoiding Catalan structures. For readers familiar with elimination trees, we mention that when the underlying graph is a path with vertices labeled 1, . . . , n, then its elimination trees are precisely all n-vertex binary trees. Very recently, another application of the aforementioned generation framework to derive Gray codes for geometric Catalan structures, specifically staircases and squares, has been presented in [DEHW23].

1.3. Outline of this paper. In Section 2 we introduce basic notions 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 classes of binary trees that are characterized by pattern avoidance. In Section 5 we establish the equality between certain tree patterns that differ in few contiguous or non-contiguous edges. In Section 6 we report on our computational results on counting pattern-avoiding binary trees for all tree patterns on at most 5 vertices. In Section 7 we prove bijections between different classes of pattern-avoiding binary trees and other combinatorial objects, in particular pattern-avoiding lattice paths and set partitions. In Section 8 we present results for establishing Wilf-equivalence between tree patterns. We conclude with some open problems in Section 9.

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. We consider binary trees whose vertex set is a set of consecutive integers \{i, i + 1, . . . , j\}. In particular, we write \(T_n\) for the set of binary trees with the vertex set \([n] := \{1, 2, . . . , n\}\). 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 \(\varepsilon\), so \(T_0 = \{\varepsilon\}\). 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 \(T \in T_n\) with a permutation \(\tau(T)\) of \([n]\) defined by

\[
\tau(T) := (r(T), \tau(L(T)), \tau(R(T))),
\]
where the base case of the empty tree \( \varepsilon \) 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 \( \tau(T) = \text{id}_n \), the identity permutation.

For any vertex \( i \) we let \( \beta_L(i) \) and \( \beta_R(i) \) denote the number of vertices on the left branch or right branch, respectively, starting at \( i \), with the special cases \( \beta_L(\varepsilon) := 0 \) and \( \beta_R(\varepsilon) := 0 \). We also define \( B_L(i) := \{\beta_L^{-1}(i) \mid j = 1, \ldots, \beta_L(i)\} \) and \( B_R(i) := \{\beta_R^{-1}(i) \mid j = 1, \ldots, \beta_R(i)\} \) as the corresponding sets of vertices on this branch. Lastly, we define \( B_R^-(i) := B_R(i) \setminus \beta_R^{-1}(i) \), i.e., all vertices on the right 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 [Row10] and [DPTW12] (recall Figure 1). This definition is illustrated in Figure 4. A tree pattern is a pair \( (P, e) \) where \( P \in \mathcal{T}_k \) and \( e : [k] \setminus r(P) \to \{0, 1\} \). 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 \( f : [k] \to [n] \) satisfying the following conditions:

![Figure 4. Illustration of definitions related to binary trees.](image)

![Figure 5. Illustration of our notion of pattern containment in binary trees.](image)
(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 [Row10] and [DPTW12] 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.

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

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

Note that \(\mathcal{T}_0(P, e) = \{\varepsilon\}\) for any nonempty tree pattern \((P, e)\). For avoiding multiple patterns \((P_1, e_1), \ldots, (P_\ell, e_\ell)\) simultaneously, we define

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

Clearly, the set of binary trees that avoids a tree pattern \((P, e), P \in \mathcal{T}_k\), is monotonously non-decreasing in \(e\), i.e., if \(e(i) \leq e'(i)\) for every vertex \(i \in [k] \setminus r(P)\), then \(\mathcal{T}_n(P, e) \subseteq \mathcal{T}_n(P, e')\).

Given a tree pattern \((P, e)\) and a vertex \(i\) in \(P\), we sometimes consider the induced subpattern \((P(i), e_P(i))\), where \(e_P(i)\) denotes the restriction of \(e\) to the vertex set of \(P(i) \setminus i\).

We often write a tree pattern \((P, e), P \in \mathcal{T}_k\), in compact form as a pair \((\tau(P), (e(\tau_2), \ldots, e(\tau_k)))\) 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 \(\mathcal{T}_n(\mu(P, e)) = \mathcal{T}_n(P, e)\), in particular \((P, e)\) and \(\mu(P, e)\) are Wilf-equivalent.

\[
\mu(P, e) = (421358679, 01110101)
\]

\[
\tau(P) = 652143879
\]

\[
e = (e(5), e(2), e(1), e(4), e(3), e(8), e(7), e(9)) = 10110011
\]

\[
(P, e) = (652143879, 10110011)
\]

Figure 6. Compact encoding of binary tree patterns and mirroring operation.
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 permutation mesh pattern in 231-avoiding permutations (Theorem 2.1 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 $\pi$ contains $\tau$ 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 $\pi$ does not contain $\tau$, then we say that $\pi$ avoids $\tau$. We write $S_n(\tau)$ for the permutations from $S_n$ that avoid the pattern $\tau$. 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., [Kno77]).

**Lemma 1.** The mapping $\tau : T_n \to S_n(231)$ defined in (1) is a bijection.

3.2. Mesh patterns. Mesh patterns were introduced by Brändén and Claesson [BC11], and they generalize classical permutation patterns discussed in the previous section. We recall 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 $\pi$.

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 $\sigma$ by the grid representation of $\tau$, 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 $\tau$ in $\pi$. 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 $(\tau, \emptyset)$.

![Figure 7. Illustration of mesh pattern containment.](image-url)
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 \(\tau(P) \in S_k\), which we abbreviate to \(\rho := \tau(P)^{-1} \in S_k\). The permutation \(\rho\) 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

\[ C_i := \{(\rho(i) - 1, j) | j \in B_R^{-}(i)\}, \quad (2a) \]

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

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

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

\[ \sigma(P,e) := \left(\tau(P), \bigcup_{i \in [k]} C_i \cup \bigcup_{i \in [k] \setminus r(P)} C'_i\right). \quad (2c) \]

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 \(P(i)\).

The following generalization of Lemma 1 is the main result of this section. Our theorem also generalizes Theorem 12 from [PSSS14], 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 \(\tau: \mathcal{T}_n(P,e) \to S_n(231, \sigma(P,e))\) is a bijection.
This theorem extends naturally to avoiding multiple tree patterns \((P_1, e_1), \ldots, (P_\ell, e_\ell)\), i.e., \(\tau: T_n((P_1, e_1), \ldots, (P_\ell, e_\ell)) \rightarrow S_n(231, \sigma(P_1, e_1), \ldots, \sigma(P_\ell, e_\ell))\) is a bijection.

Proof. By Lemma 1, \(\tau: T_n \rightarrow S_n(231)\) is a bijection. Consequently, it suffices to show that \(T \in T_n\) contains the tree pattern \((P, e)\) if and only if \(\tau(T) \in S_n(231)\) contains the mesh pattern \(\sigma(P, e)\).

Using the definitions of tree patterns and mesh patterns from Sections 2.2 and 3.2, respectively, and combining them with 2, a straightforward induction shows that if \(T \in T_n\) contains the tree pattern \((P, e)\), then \(\tau(T)\) contains the mesh pattern \(\sigma(P, e)\). For this argument we also use that in the mesh pattern \(\sigma(P, e) = (\tau(P), C)\), none of the four corner cells is shaded, i.e., we have \((0,0), (0,k), (k,0), (k,k) \notin C\).

It remains to show that if \(\tau(T)\) for \(T \in T_n\) contains the mesh pattern \(\sigma(P, e) = (\tau(P), C)\), then \(T\) contains the tree pattern \((P, e)\). This means that there are indices \(\nu_1 < \cdots < \nu_k\) satisfying conditions (i) and (ii) stated in Section 3.2. We define the abbreviation \(\pi := \tau(T)\). Let \(\lambda_1 < \cdots < \lambda_k\) be the values \(\pi(\nu_1), \ldots, \pi(\nu_k)\) sorted in increasing order, and let \(Q := \{\lambda_i \mid i \in [k]\}\) be the corresponding set of values of \(\tau(T)\) that correspond to this occurrence of the mesh pattern \(\sigma(P, e)\). From 1, we have \(\tau(T) = (\tau(T), \tau(L(T)), \tau(R(T))) \in S_n(231)\), and so we consider the following four cases, illustrated in Figure 10.

Case (a): \(Q \subseteq \tau(L(T))\). In this case \(\tau(L(T))\) contains the mesh pattern \(\sigma(P, e)\). It follows by induction that \(L(T)\) contains the tree pattern \((P, e)\), and therefore \(T\) contains the tree pattern \((P, e)\).
Case (b): $Q \subseteq \tau(R(T))$. In this case $\tau(R(T))$ contains the mesh pattern $\sigma(P,e)$. It follows by induction that $R(T)$ contains the tree pattern $(P,e)$, and therefore $T$ contains the tree pattern $(P,e)$.

Case (c): $r(T) \in Q$. We define $a := c_L(r(P)) = r(L(P))$ and $b := c_R(r(P)) = r(R(P))$. We assume that $a, b \neq \varepsilon$; the other cases are analogous. In this case $\tau(L(T))$ contains the mesh pattern $\sigma(L(P), e_{L(P)})$ and $\tau(R(T))$ contains the mesh pattern $\sigma(R(P), e_{R(P)})$.

We define $\rho := \tau(P)^{-1} \in S_k$. From (2a) we see that if $c_R(a) \neq \varepsilon$, then we have $(\rho(a) - 1, j) = (1, j) \in C$ for all $j \in B^R_T(a)$. Furthermore, if $e(a) = 1$, then we have $(\rho(a) - 1, \min P(a) - 1) = (1, 0) \in C$ and $(\rho(a) - 1, \max P(a)) = (1, r(P) - 1) \in C$. We thus obtain that if $e(a) = 1$, then the occurrence of the mesh pattern $\sigma(L(P), e_{L(P)})$ in $\tau(L(T))$ must contain the first element.
of \( \tau(L(T)) \). By induction, we obtain that \( L(T) \) contains the tree pattern \((L(P), e_{L(P)})\), and if \( e(a) = 1 \) then an occurrence of this pattern includes the vertex \( c_L(T) \).

Similarly, from (2a) we see that if \( c_R(b) \neq \varepsilon \), then we have \((\rho(b) - 1, j) \in C \) for all \( j \in B_R(b) \). Furthermore, if \( e(b) = 1 \), then we have \((\rho(b) - 1, \min P(b) - 1) = (\rho(b) - 1, r(P)) \in C \) and \((\rho(b) - 1, \max P(b)) = (\rho(b) - 1, k) \in C \). We thus obtain that if \( e(b) = 1 \), then the occurrence of the mesh pattern \( \sigma(R(P), e_{R(P)}) \) in \( \tau(R(T)) \) must contain the first element of \( \tau(R(T)) \). By induction, we obtain that \( R(T) \) contains the tree pattern \((R(P), e_{R(P)})\), and if \( e(b) = 1 \) then an occurrence of this pattern includes the vertex \( c_R(T) \).

Combining these observations yields that \( T \) contains the tree pattern \((P, e)\).

**Case (d):** \( r(T) \notin Q, Q \cap \tau(L(T)) \neq \emptyset \) and \( Q \cap \tau(R(T)) \neq \emptyset \).

We define \( \rho := \tau(P)^{-1} \in S_k, b := \beta_R(r(P)) \) and \( r_i := c_R^{-1}(r(P)) \) for \( i = 1, \ldots, b \). For any vertex \( i \in [k] \) of \( P \), we have that all \( j \in L(i) \) come after \( i \) in \( \tau(P) \) and are smaller than \( i \). Consequently, there is an integer \( 1 \leq j < b \) such that for all \( i = 1, \ldots, j \) we have \( \lambda_i \in \tau(L(T)) \) and \( \lambda_k \in \tau(L(T)) \) for all \( k \in L(r_i) \), and moreover for all \( i = j + 1, \ldots, b \) we have \( \lambda_i \in \tau(R(T)) \) and \( \lambda_k \in \tau(R(T)) \) for all \( k \in L(r_i) \). However, by the definition (2a) we have \((0, r_j) \in C \), which implies that \((1, r(T)) \in R_0 r_i \), so this case cannot occur.

This completes the proof of the theorem. \( \square \)

## 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 10, respectively).

### 4.1. Tree rotations and slides

A natural and well-studied operation on binary trees is tree rotations; see Figure 2. We consider a tree \( T \in T_n \) and one of its edges \((i, j)\) with \( j = c_L(i) \), 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 \( \mathcal{L}_n \subseteq T_n \). 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 \( \mathcal{L}_n \subseteq T_n \) 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.

\[(P, e) = \begin{array}{c}
\text{not visited:} \\
4△ 1234 \\
3△ 4123 \\
2△ 4312 \\
1△ 3124 \\
\end{array} \]

\[T_1 \rightarrow (4\Delta)^3 T_2 \rightarrow (3\Delta)^2 T_3 \rightarrow 4\downarrow T_4 \rightarrow 2\Delta T_5 \rightarrow 4\Delta T_6 \rightarrow 3\downarrow T_7 \rightarrow (4\downarrow)^2 T_8 \]

**Figure 11.** Run of Algorithm S that visits all binary trees in the set \(T_4(P, e)\). Below each tree \(T\) is the corresponding permutation \(\tau(T)\).

\[(P, e) = \begin{array}{c}
\text{not visited:} \\
4\Delta 1234 \\
3\Delta 4123 \\
2\Delta 4312 \\
1\Delta 3124 \\
\end{array} \]

\[T_1 \rightarrow 4\Delta T_2 \rightarrow 4\Delta T_3 \rightarrow 4\Delta T_4 \rightarrow 3\Delta T_5 \rightarrow 4\downarrow T_6 \]

**Figure 12.** Run of Algorithm S that does not visit all binary trees in the set \(T_4(P, e)\).

To illustrate the algorithm, consider the example in Figure 11. 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 \(T_4(P, e)\).

Now consider the example in Figure 12 where the algorithm terminates after having visited six different trees from \(T_4(P, e)\). However, the set \(T_4(P, e)\) contains two more trees that are not visited by the algorithm.

We now formulate simple sufficient conditions on the tree pattern \((P, e)\) ensuring that Algorithm S successfully visits all trees in \(T_n(P, e)\). Specifically, we say that a tree pattern \((P, e), P \in T_k\), is **friendly**, if it satisfies the following three conditions; see Figure 13.

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

(ii) For every \(j \in B_{\bar{R}}(r(P)) \setminus r(P)\) 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 \(e(cL(k)) = 0\), i.e., if the edge from \(k\) to its parent is contiguous, then the edge to its left child must be non-contiguous.

**Figure 13.** Definition of friendly tree patterns.
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 \((P_1, e_1), \ldots, (P_t, e_t)\) be friendly tree patterns. Then Algorithm S initialized with the tree \( \tau^{-1}(\text{id}_n) \) visits every binary tree from \( T_n((P_1, e_1), \ldots, (P_t, e_t)) \) exactly once.

Recall that \( \tau^{-1}(\text{id}_n) \) is the right path, i.e., the tree that corresponds to the identity tree. Note that by condition (i) in the definition of friendly tree pattern, we have \( \tau^{-1}(\text{id}_n) \in T_n((P_1, e_1), \ldots, (P_t, e_t)) \). We shall see that our notion of friendly tree patterns is inherited from the notion of tame mesh permutation patterns used in [HHMW22, Thm. 15].

### 4.3. Permutation languages.

We prove Theorem 3 by applying the Hartung–Hoang–Mütze–Williams generation framework [HHMW22]. Let us recap the most important concepts. We interpret a permutation \( \pi \in S_n \) in one-line notation as a string as \( \pi = \pi(1), \pi(2), \ldots, \pi(n) = a_1a_2\cdots a_n \). Recall that \( \varepsilon \in S_0 \) denotes the empty permutation. For any \( \pi \in S_{n-1} \) and any \( 1 \leq i \leq n \), we write \( c_i(\pi) \in S_n \) for the permutation obtained from \( \pi \) by inserting the new largest value \( n \) at position \( i \) of \( \pi \), i.e., if \( \pi = a_1\cdots a_{n-1} \) then \( c_i(\pi) = a_1\cdots a_{i-1}n a_i\cdots a_{n-1} \). Moreover, for \( \pi \in S_n \), we write \( p(\pi) \in S_{n-1} \) for the permutation obtained from \( \pi \) by removing the largest entry \( n \). Given a permutation \( \pi = a_1\cdots a_n \) with a substring \( a_i\cdots a_j \) with \( a_i > a_{i+1}, \ldots, a_j \), a **right jump of the value** \( a_i \) **by** \( j - i \) **steps** is a cyclic left rotation of this substring by one position to \( a_{i+1}\cdots a_ja_i \). Similarly, given a substring \( a_i\cdots a_j \) with \( a_j > a_i, \ldots, a_{j-1} \), a **left jump of the value** \( a_j \) **by** \( j - i \) **steps** is a cyclic right rotation of this substring to \( a_ja_i\cdots a_{j-1} \).

The framework from [HHMW22] uses the following simple greedy algorithm to generate a set of permutations \( L_n \subseteq S_n \). We say that a jump is **minimal** (w.r.t. \( L_n \)), if every jump of the same value in the same direction by fewer steps creates a permutation that is not in \( L_n \).

**Algorithm J** (Greedy minimal jumps). This algorithm attempts to greedily generate a set of permutations \( L_n \subseteq S_n \) using minimal jumps starting from an initial permutation \( \pi_0 \in L_n \).

1. **[Initialize]** Visit the initial permutation \( \pi_0 \).
2. **[Jump]** Generate an unvisited permutation from \( L_n \) by performing a minimal jump of the largest possible value in the most recently visited permutation. If no such jump exists, or the jump direction is ambiguous, then terminate. Otherwise visit this permutation and repeat J2.

The following main result from [HHMW22] provides a sufficient condition on the set \( L_n \) to guarantee that Algorithm J successfully generates all permutations in \( L_n \). This condition is captured by the following closure property of the set \( L_n \). A set of permutations \( L_n \subseteq S_n \) is called a **zigzag language**, if either \( n = 0 \) and \( L_0 = \{\varepsilon\} \), or if \( n \geq 1 \) and \( L_{n-1} := \{p(\pi) \mid \pi \in L_n\} \) is a zigzag language such that for every \( \pi \in L_{n-1} \) we have \( c_1(\pi) \in L_n \) and \( c_n(\pi) \in L_n \).

**Theorem 4** ([HHMW22]). Given any zigzag language of permutations \( L_n \) and initial permutation \( \pi_0 = \text{id}_n \), Algorithm J visits every permutation from \( L_n \) exactly once.

### 4.4. Tame permutation patterns and friendly tree patterns.

We say that an infinite sequence \( L_0, L_1, \ldots \) of sets of permutations is **hereditary**, if \( L_i = p(L_i) \) holds for all \( i \geq 1 \). We say that a (classical or mesh) permutation pattern \( \tau \) is **tame**, if \( S_n(\tau) \), \( n \geq 0 \), is a hereditary sequence of zigzag languages.

**Lemma 5** ([HHMW22, Lem. 6]). Let \( L_0, L_1, \ldots \) and \( M_0, M_1, \ldots \) be two hereditary sequences of zigzag languages. Then \( L_n \cap M_n \) for \( n \geq 0 \) is also a hereditary sequence of zigzag languages.
The following result about classical permutation patterns was proved in \[\text{HHMW22}\].

**Lemma 6** \[\text{HHMW22} \text{ Lem. 9}\]. *If a pattern \(\tau \in S_k, k \geq 3\), does not have the largest value \(k\) at the leftmost or rightmost position, then it is tame.*

**Lemma 7** \[\text{HHMW22} \text{ Thm. 15}\]. *Let \(\sigma = (\tau, C)\), \(\tau \in S_k, k \geq 3\), be a mesh pattern, and let \(i\) be the position of the largest value \(k\) in \(\tau\). If the pattern satisfies the following four conditions, then it is tame:

(i) \(i\) is different from 1 and \(k\).

(ii) For all \(a \in \{0, \ldots, k\} \setminus \{i-1, i\}\), we have \((a, k) \notin C\).

(iii) If \((i-1, k) \in C\), then for all \(a \in \{0, \ldots, k\} \setminus i-1\) we have \((a, k-1) \notin C\) and for all \(b \in \{0, \ldots, k-2\}\) we have that \((i, b) \in C\) implies \((i-1, b) \in C\).

(iv) If \((i, k) \in C\), then for all \(a \in \{0, \ldots, k\} \setminus i\) we have \((a, k-1) \notin C\) and for all \(b \in \{0, \ldots, k-2\}\) we have that \((i-1, b) \in C\) implies \((i, b) \in C\).*

The following crucial lemma connects friendliness of tree patterns to tameness of mesh patterns.

**Lemma 8.** *Let \(P, e\), \(P \in \mathcal{F}_k\), be a friendly tree pattern. Consider the mesh pattern \(\sigma(P, e) = (\tau(P), C)\) defined in (2), let \(i\) be the position of the largest value \(k\) in \(\tau(P)\), and define the mesh pattern \(\sigma^-(P, e) := (\tau(P), C \setminus \{i-1, k\})\). Then the mesh pattern \(\sigma^-(P, e)\) is tame.*

**Proof.** By condition (i) of friendly mesh patterns, we have \(p(k) \neq \varepsilon\) and \(c_L(k) \neq \varepsilon\) and therefore \(i > 1\) and \(i < k\), respectively, which implies that condition (i) of Lemma \[\text{7}\] is satisfied for both \(\sigma(P, e)\) and \(\sigma^-(P, e)\).

By condition (ii) of friendly mesh patterns and the definition \[\text{(2)}\], we have \((a, k) \notin C\) for all \(a \in \{0, \ldots, k\} \setminus i-1\). Furthermore, we have \((i-1, k) \in C\) if and only if \(e(k) = 1\). It follows that conditions (ii) and (iv) of Lemma \[\text{7}\] are satisfied for both \(\sigma(P, e)\) and \(\sigma^-(P, e)\). Furthermore, if \(e(k) = 0\) then condition (iii) is also satisfied for both \(\sigma(P, e)\) and \(\sigma^-(P, e)\). Lastly, if \(e(k) = 1\) then we have \((i-1, k) \in C\), so \(\sigma(P, e)\) does not satisfy condition (iii), but \(\sigma^-(P, e)\) does. \[\square\]

**Lemma 9.** *The mesh patterns \(\sigma(P, e)\) and \(\sigma^-(P, e)\) defined in Lemma \[\text{8}\] satisfy \(S_n(231, \sigma(P, e)) = S_n(231, \sigma^-(P, e))\).*

**Proof.** It suffices to show that a 231-avoiding permutation \(\pi \in S_n\) that contains an occurrence of \(\sigma(P, e)\) also contains an occurrence of \(\sigma^-(P, e)\). This proof uses an exchange argument; see Figure \[14\]. Let \(i\) be the position of the largest value \(k\) in \(\tau(P)\). Furthermore, for any \(j \in [k]\) we let \(q(j)\) denote the point in \(G(\pi)\) corresponding to the value \(j\) in the occurrence of \(\sigma(P, e)\) in \(\pi\). If \(R_{i-1,k}\) contains no points from \(G(\pi)\), then this is also an occurrence of \(\sigma^-(P, e)\), and we are done. Otherwise, we let \(q'\) be the leftmost highest point in \(G(\pi) \cap R_{i-1,k}\), and we claim that replacing \(q(k)\) by \(q'\) creates an occurrence of \(\sigma^-(P, e)\) in \(\pi\). To verify this we first observe that \((i, k-1) \notin C\) by condition (iii) of friendly mesh patterns. Secondly, if \((a-1, k-1) \in C\) for some \(a \in L(k) \setminus c_L(k)\), then by \[\text{(2)}\] we have \((a, k-1) \in G(\tau(P))\). This implies that \(R_{a-1,k} \cap G(\pi) = \emptyset\), otherwise a point in this region would form an occurrence of 231 together with the points \(q(k)\) and \(q(k-1)\). Thirdly, if \((i, a) \in C\) for some \(a \in L(k)\), then by \[\text{(2)}\] we have \(a \in B_R(c_L(k))\). This implies that \(R_{i-1,a} \cap G(\pi) = \emptyset\), otherwise a point in this region would form an occurrence of 231 together with the points \(q(k)\) and \(q(a)\). This completes the proof. \[\square\]
4.5. Proof of Theorem 3. We are now in position to prove Theorem 3.

Proof of Theorem 3. Lemma 6 shows that the classical pattern 231 is tame. As $(P_i, e_i)$, $i \in \ell$, are friendly tree patterns, the mesh patterns $\sigma^-(P_i, e_i)$, $i \in \ell$, are tame by Lemma 8. Combining Lemma 5 and Lemma 9 yields that

$$L_n := S_n(231, \sigma^-(P_1, e_1), \ldots, \sigma^-(P_\ell, e_\ell))$$

for $n \geq 0$ is a hereditary sequence of zigzag languages. Theorem 4 thus guarantees that Algorithm J initialized with the identity permutation $id_n$ visits every permutation of $L_n$ exactly once. We now show that Algorithm S is the preimage of Algorithm J under the mapping $\tau$, which is a bijection between $L_n := T_n((P_1, e_1), \ldots, (P_\ell, e_\ell))$ and $L_n$ by Theorem 2. It was shown in [HHMW22] Sec. 3.3 that minimal jumps in $S_n(231)$ are in one-to-one correspondence with tree rotations in $T_n$. Specifically, a minimal left jump of a value $j$ in the permutation corresponds to an up-rotation of $j$ in the binary tree, and a minimum right jump of $j$ corresponds to a down-rotation of $j$. Consequently, minimal jumps in $L_n$ are in one-to-one correspondence with minimal slides in $L_n$. This completes the proof of the theorem. □

4.6. 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 below is a straightforward translation of the history-free Algorithm M for zigzag languages presented in [MM23] 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 $T_n$ that avoid $(P_1, e_1), \ldots, (P_\ell, e_\ell)$, i.e., the set $L_n := T_n((P_1, e_1), \ldots, (P_\ell, e_\ell)) \subseteq T_n$ 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 $T \leftarrow \tau^{-1}(id_n)$, and $o_j \leftarrow \Delta, s_j \leftarrow j$ for $j = 1, \ldots, n$.
H2. [Visit] Visit the current binary tree $T$. 

Figure 14. Exchange argument in the proof of Lemma 9.
**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 \( o_j = \Delta \) and down if \( o_j = \nabla \).

**H5.** [Update o and s] Set \( s_n \leftarrow n \). If \( o_j = \Delta \) and \( j \) is either the root or its parent is larger than \( j \) set \( o_j = \nabla \), or if \( o_j = \nabla \) and \( j \) has no left child set \( o_j = \Delta \), 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 \((P_i, e_i)\), \( i \in \{0, \ldots, \ell\} \), in \( T \).

**Theorem 10.** Let \((P_1, e_1), \ldots, (P_\ell, e_\ell)\) be friendly tree patterns with \( P_i \in \mathcal{T}_{k_i} \) for \( i \in \{1, \ldots, \ell\} \). 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 \( O(n^2 \sum_{i=1}^{\ell} k_i^2) \) per binary tree.

**Proof.** The correctness of Algorithm H follows from [MM23, Thm. 29].

For the running time, note that any slide consists of at most \( n \) rotations, and that testing whether \( T \in \mathcal{T}_n \) contains the tree pattern \((P_i, e_i)\), \( P_i \in \mathcal{T}_{k_i} \), can be done in time \( O(n k_i^2) \) by dynamic programming as follows. We store for each vertex of \( T \) a table of size \( k_i \) that gives information for each vertex of \( P_i \) whether the corresponding subtree of \( T \) contains the corresponding subtree of \( P_i \) as a pattern. In fact, we need two such tables, one for tracking ‘containment’ and the other for tracking the stronger property ‘containment at the root’. This information can be computed bottom-up in time \( O(k_i^2) \) for each of the \( n \) vertices of \( T \) (cf. [HO82]).

For details, see our C++ implementation [cos].

---

5. **Equality of tree patterns**

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 \((e(i) = 0)\). The following theorem captures these conditions formally, and it establishes that \( \mathcal{T}_n(P, e) = \mathcal{T}_n(P, e') \) for tree patterns \((P, e)\) and \((P, e')\) where \( e \) and \( e' \) differ only in a single value. Theorem [11] will be used heavily in the tables in the next section, where those ‘don’t care’ values of \( e \) are denoted by the hyphen \(-\). The statement and proof of this theorem is admittedly slightly technical, and we recommend to skip it on first reading.

Let \((P, e), P \in \mathcal{T}_k\), be a tree pattern. For any vertex \( i \in [k] \), we define

\[
B^j_L(i) := \{ c^j_L(i) \mid \ell \geq 0 \text{ and } e(c^j_L(i)) = 1 \text{ for all } j = 1, \ldots, \ell \},
\]

\[
A_R(i) := \{ a \in [k] \mid i = c^j_R(a) \text{ for some } \ell \geq 0 \text{ and } e(c^j_R(a)) = 1 \text{ for all } j = 1, \ldots, \ell \}. \tag{3a}
\]

In words, \( B^j_L(i) \) are the descendants of \( i \) in \( P \) that are reachable from \( i \) along a path of contiguous left edges. Furthermore, \( A_R(i) \) are the predecessors of \( i \) in \( P \) that reach \( i \) along a path of contiguous right edges. Both sets include the vertex \( i \) itself. We also write \( a_R(i) \) for the top vertex from \( A_R(i) \) in the tree \( P \).
We define analogous sets $B'_R(i)$ and $A_L(i)$ and vertices $a_L(i)$ that are obtained by interchanging left and right in the definitions before. Specifically, these sets are defined as

$$B'_R(i) := \{ c_{R}^{\ell}(i) | \ell \geq 0 \text{ and } e(c_{R}^{\ell}(i)) = 1 \text{ for all } j = 1, \ldots, \ell \},$$

$$A_L(i) := \{ a \in [k] | i = c_{L}(a) \text{ for some } \ell \geq 0 \text{ and } e(c_{L}^{\ell}(a)) = 1 \text{ for all } j = 1, \ldots, \ell \},$$

and $a_L(i)$ is defined as the top vertex from $A_L(i)$ in the tree $P$. Using these definitions, we consider the following subsets $X, Y, Y' \subseteq [k]$ of vertices of $P$; see Figure 15

$$X := \{ i \in [k] | c_{L}(i) = \varepsilon \land c_{R}(i) = \varepsilon \},$$

$$Y := \{ i \in [k] | c_{L}(i) \neq \varepsilon \land i = c_{R}(p(i)) \land (A_{R}(p(i)) = \{ p(i) \} \lor B'_L(i) = \{ i \}) \land (c_{L}(j) = \varepsilon \text{ for all } j \in A_{R}(p(i)) \land (c_{R}(j) = \varepsilon \text{ for all } j \in B'_L(i)) \land (a_{R}(p(i)) = r(P) \lor e(a_{R}(p(i))) = 0) \},$$

$$Y' := \{ i \in [k] | c_{R}(i) \neq \varepsilon \land i = c_{L}(p(i)) \land (A_{L}(p(i)) = \{ p(i) \} \lor B'_R(i) = \{ i \}) \land (c_{R}(j) = \varepsilon \text{ for all } j \in A_{L}(p(i)) \land (c_{L}(j) = \varepsilon \text{ for all } j \in B'_R(i)) \land (a_{L}(p(i)) = r(P) \lor e(a_{L}(p(i))) = 0) \}.$$

Note that $X$ is simply the set of all leaves of $P$. The six conditions in the conjunction that defines $Y$ express the following facts: (1) $i$ has a left child; (2) $i$ is a right child of its parent $p(i)$; (3) one of the sets $A_{R}(p(i))$ or $B'_L(i)$ is trivial; (4) no vertex in $A_{R}(p(i))$ has a left child; (5) no vertex in $B'_L(i)$ (including $i$ itself) has a right child; (6) the top vertex $a_{R}(p(i))$ in $A_{R}(p(i))$ is either the root or the edge to its parent is non-contiguous. The definition of $Y'$ is analogous, but with left and right interchanged.

**Theorem 11.** Let $(P, e), P \in \mathcal{T}_k$, be a tree pattern, and let $X, Y, Y'$ be the sets of vertices in $P$ defined in (3) w.r.t. $(P, e)$. Furthermore, let $i \in X \cup Y \cup Y'$ be a vertex of $P$ with $e(i) = 0$, and
define $e': [k] \setminus r(P) \rightarrow \{0, 1\}$ by $e'(i) := 1$ and $e'(j) := e(j)$ for all $j \in [k] \setminus i$. Then we have $T_n(P, e) = T_n(P, e')$.

Note that $(P, e')$ differs from $(P, e)$ in that the edge $(i, p(i))$ from $i$ to its parent $p(i)$ is contiguous instead of non-contiguous.

**Proof.** It suffices to show that if $T \in T_n$ contains $(P, e)$, then $T$ contains $(P, e')$. Let $T \in T_n$ and consider an occurrence of $(P, e)$ in $T$, witnessed by an injection $f: [k] \rightarrow [n]$ that satisfies the conditions stated in Section 2.2.

We consider the cases $i \in X$, $i \in Y$, or $i \in Y'$ separately. The last two are symmetric, so it suffices to consider whether $i \in X$ or $i \in Y$.

**Case (a):** $i \in X$. As $e(i) = 0$, $f(i)$ is a descendant of $f(p(i))$ in $T$. Instead of mapping $i$ to $f(i)$ in $T$, we remap it to the corresponding direct child of $f(p(i))$ in $T$. Specifically, if $i = c_L(p(i))$ in $P$, then we remap $i$ to $c_L(f(p(i)))$ in $T$, and if $i = c_R(p(i))$ in $P$, then we remap $i$ to $c_R(f(p(i)))$ in $T$. This is possible as $i$ is a leaf in $P$. This shows that $T$ contains $(P, e')$, as claimed.

**Case (b):** $i \in Y$. We distinguish the subcases $A_R(p(i)) = \{p(i)\}$ and $B'_L(i) = \{i\}$, at least one of which must hold by the definition of $Y$ in (3c). The arguments in these two cases are illustrated in Figure 16.

**Case (b1):** $A_R(p(i)) = \{p(i)\}$, i.e., $p(i)$ is the root or the edge from $p(i)$ to its parent is non-contiguous. We consider the vertex $f(i)$ in $T$, and we let $b$ be the top vertex in $T$ such that $f(i) \in B_L(b)$. We remap $p(i)$ to $p(b)$ in $T$, and we remap $B'_L(i)$ (including $i$ itself) to $b$ and its direct left descendants. By the definition (3c), we know that in $P$ we have $c_L(p(i)) = \varepsilon$, $c_R(j) = \varepsilon$ for all $j \in B'_L(i)$, and $p(i) = r(P)$ or $e(p(i)) = 0$, which ensures that the remapping indeed witnesses an occurrence of $(P, e')$.

**Case (b2):** $B'_L(i) = \{i\}$, i.e., the edge from $i$ to its left child is non-contiguous. We consider the vertex $f(p(i))$ in $T$, and we let $b$ be the vertex in $T$ such that $b \in B_R(f(p(i)))$ and $f(i) \in L(b)$.

**Figure 16.** Illustration of the proof of Theorem 11. The dashed arrows indicate the remapping argument.
We remap $A_R(p(i))$ (including $p(i)$ itself) to $p(b)$ and its direct ancestors, and we remap $f(i)$ to $b$. By the definition (3c), we know that in $P$ we have $c_L(j) = \varepsilon$ for all $j \in A_R(p(i))$, $c_R(i) = \varepsilon$, and $a_R(p(i)) = \tau(P)$ or $e(a_R(p(i))) = 0$, which ensures that the remapping indeed witnesses an occurrence of $(P,e')$. \hfill $\square$

6. Tree patterns on at most 5 vertices

In order to mine interesting conjectures about tree pattern avoidance[1] 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 $|T_n(P,e)|$ for $n = 1, \ldots, 12$, and searched for matches within the OEIS [oeis24]. 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. Some of the $e$-sequences contain ‘don’t care’ entries $\_\_\_\_\_$, which means that both possible $e$-values 0 or 1 yield the same sets of pattern-avoiding trees by Theorem 11.

The last column contains a reference to a proof that the counting sequence is indeed the listed OEIS entry. The Wilf-equivalence column contains pointers to a Wilf-equivalent tree pattern in Figures 2[29,31] that has an established OEIS entry. The question mark at the pattern (31245, 1–0–) in Table 3 means that the match has not been proved formally for all $n$, but was only verified experimentally for $n \leq 12$ (so this is an open problem). For every other pattern we have a reference or a pointer to a Wilf-equivalent pattern that has a reference, unless it is one of the 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.

| Table 1. Tree patterns with 3 vertices. See Section 6 for explanations. |
|---|---|---|---|---|
| $P$ | $e$ | Friendly | Counts $|T_n(P,e)|$ for $n = 1, \ldots, 12$ | OEIS | Wilf-equivalence |
| 123-0- | 1 2 4 8 16 32 | 64 128 256 | 512 1024 2048 | A000079 | DPTW12 Thm. 1; Sec. 7.1.2 |
| 1-6 | 1 2 4 8 16 32 | 512 1024 2048 | 4096 8192 16384 | A003462 | Roe10 Class 2; Tab. 5 |
| 124-0-0- | 1 2 4 8 16 32 | 64 128 256 | 512 1024 2048 | A000108 | DPTW12 Thm. 1; Sec. 7.2.1; Tab. 4 |
| 11- | 1 2 4 8 16 32 | 512 1024 2048 | 4096 8192 16384 | A003462 | Roe10 Class 2; Tab. 5 |
| 123-0-0- | 1 2 4 8 16 32 | 64 128 256 | 512 1024 2048 | A003462 | DPTW12 Thm. 1; Tab. 4 |

| Table 2. Tree patterns with 4 vertices. |
|---|---|---|---|---|
| $P$ | $e$ | Friendly | Counts $|T_n(P,e)|$ for $n = 1, \ldots, 12$ | OEIS | Wilf-equivalence |
| 1234-0-0- | 1 2 5 13 35 97 275 794 2327 6905 20705 62642 | | | A000079 | DPTW12 Thm. 1 |
| 10- | 1 2 5 13 35 97 275 794 2327 6905 20705 62642 | | | A003462 | Roe10 Class 5.1; Tab. 4 |
| 11- | 1 2 5 13 35 97 275 794 2327 6905 20705 62642 | | | A003462 | | |
| 1234-0-0-0- | 1 2 5 13 35 97 275 794 2327 6905 20705 62642 | | | A003462 | Roe10 Class 5.1; Tab. 4 |
| 11- | 1 2 5 13 35 97 275 794 2327 6905 20705 62642 | | | A003462 | | |

1and to pretend that we are doing ‘big data’
<table>
<thead>
<tr>
<th>P</th>
<th>0</th>
<th>Counts</th>
<th>OEIS</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-</td>
<td>2</td>
<td>1 2 5 14 41 122 365 1094 3281 9842 29295 88574 ...</td>
<td>A070951</td>
<td>DPTW12 Thm. 1</td>
</tr>
<tr>
<td>0-</td>
<td>2</td>
<td>1 2 5 14 41 123 374 1147 3538 10958 34042 105997 ...</td>
<td>A053991</td>
<td>(12543, 0-1) Lem. 17</td>
</tr>
<tr>
<td>0-</td>
<td>2</td>
<td>1 2 5 14 41 123 375 1157 3603 11304 5683 13129</td>
<td>A265508</td>
<td>(12543, -11) Lem. 17</td>
</tr>
<tr>
<td>0-</td>
<td>2</td>
<td>1 2 5 14 41 123 384 1210 3665 12482 40677 133572</td>
<td>A159772</td>
<td>(12543, 10-) Lem. 17</td>
</tr>
<tr>
<td>0-</td>
<td>2</td>
<td>1 2 5 14 41 123 387 1202 3819 12255 39651 129190</td>
<td>A265509</td>
<td>(12543, 11-) Lem. 17</td>
</tr>
<tr>
<td>0-</td>
<td>2</td>
<td>1 2 5 14 41 123 388 1211 3875 12822 42309 140244</td>
<td>A159768</td>
<td>(12543, 11-) Lem. 17</td>
</tr>
<tr>
<td>0-</td>
<td>2</td>
<td>1 2 5 14 41 135 393 1205 4147 13798 64467 158170</td>
<td>A031760</td>
<td>(12543, 11-) Lem. 17</td>
</tr>
</tbody>
</table>

**Table 3. Tree patterns with 5 vertices.**
7. Bijections with other combinatorial objects

In this section we establish bijections between pattern-avoiding binary trees and other combinatorial objects, specifically binary strings, pattern-avoiding Motzkin paths, and pattern-avoiding set partitions.

7.1. Binary trees and bitstrings.

7.1.1. **Bijection between \( T_n(132, -) \) and bitstrings \( \{0, 1\}^{n-1} \).** This bijection is illustrated in Figure 17 (a). Consider a tree \( T \in T_n(P, e) \) where \( (P, e) := (132, -) \). We define \( b := \beta_L(r(T)) \) and \( \ell_i := c_{\ell_1}^{-1}(r(T)) \) for \( i = 1, \ldots, b \), i.e., we consider the left branch \( (\ell_1, \ldots, \ell_b) \) starting from the root of \( T \). Due to the forbidden tree pattern \((P, e)\), the tree \( T \) has exactly one right branch with \( \beta_R(\ell_i) \) many vertices starting at \( \ell_i \), for all \( i = 1, \ldots, b \). We map \( T \) to a bitstring of length \( n - 1 \) by concatenating sequences of 1s and 0s alternatingly, of lengths \( \beta_R(\ell_1), \beta_R(\ell_2), \ldots, \beta_R(\ell_b) \). This is clearly a bijection between \( T_n(P, e) \) and \( \{0, 1\}^{n-1} \).

7.1.2. **Bijection between \( T_n(123, 0-) \) and bitstrings \( \{0, 1\}^{n-1} \).** This bijection is illustrated in Figure 17 (b). Consider a tree \( T \in T_n(P, e) \) where \( (P, e) := (123, 0-) \). We define \( b := \beta_L(r(T)) \) and \( \ell_i := c_{\ell_1}^{-1}(r(T)) \) for \( i = 1, \ldots, b \), i.e., we consider the left branch \( (\ell_1, \ldots, \ell_b) \) starting from the root of \( T \). Due to the forbidden tree pattern \((P, e)\), the right subtree of \( \ell_i \) is an all left-branch with \( \beta_L(c_{\ell_1}(\ell_i)) \) many vertices, for all \( i = 1, \ldots, b \). We map \( T \) to a bitstring of length \( n - 1 \) by concatenating sequences of 1s and 0s alternatingly, of lengths \( \beta_L(c_{\ell_1}(\ell_i)), \beta_L(c_{\ell_2}(\ell_i)) + 1, \ldots, \beta_L(c_{\ell_b}(\ell_i)) + 1 \). This is clearly a bijection between \( T_n(P, e) \) and \( \{0, 1\}^{n-1} \).

7.1.3. **Bijection between \( T_n(213, -) \) and bitstrings \( \{0, 1\}^{n-1} \).** This bijection is illustrated in Figure 17 (c). Consider a tree \( T \in T_n(P, e) \) where \( (P, e) := (213, -) \). Due to the forbidden tree pattern \((P, e)\), no vertex of \( T \) has two children, i.e., \( T \) is a path. We map \( T \) to a bitstring of length \( n - 1 \) by going down the path starting at the root and recording a 1-bit for every edge going to the right, and a 0-bit for every edge going to the left. This is clearly a bijection between \( T_n(P, e) \) and \( \{0, 1\}^{n-1} \).

7.2. Binary trees and Motzkin paths. In this section, we present bijections between pattern-avoiding binary trees and different types of Motzkin paths.

Specifically, we consider lattice paths with steps \( U := (1, 1) \), \( D := (1, -1) \), \( F := (1, 0) \), and \( D_h := (1, -h) \) 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 \( M_n \) 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 \( L_n \) for the set of all \( n \)-step Motzkin left
facts (OEIS A005773). An \textit{n-step Motzkin path with catastrophes} \cite{BK21} starts at \((0, 0)\), ends at \((n, 0)\), uses only steps \(\U, \D, \F, \text{ or } \D_h\) for \(h \geq 2\), such that all \(\D_h\)-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.

7.2.1. \textit{Bijection between} \(\mathcal{T}_n(123, 1-)\) \textit{and} Motzkin \textit{paths} \(\mathcal{M}_n\). This bijection is illustrated in Figure \ref{fig:tree_bitstring} (a). Consider a tree \(T \in \mathcal{T}_n(P, e)\) where \((P, e) := (123, 1-)\). 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 \(\mathcal{M}_n\).

We remark that Rowland \cite{Row10} described a bijection between \(\mathcal{T}_n(123, 1-)\) and \(\mathcal{M}_n\) that is different from \(f\).

7.2.2. \textit{Bijection between} \(\mathcal{T}_n(1432, -1-)\) \textit{and} Motzkin \textit{left factors} \(\mathcal{L}_{n-1}\). This bijection is illustrated in Figure \ref{fig:tree_bitstring} (b), and it uses as a building block the bijection \(f\) defined in the previous section. Instead of \((1432, -1-)\), we consider the mirrored tree pattern \((P, e) := \mu(1432, -1-) = (4123, -1-)\) for convenience. Consider a tree \(T \in \mathcal{T}_n(P, e)\). We define \(b := \beta_R(r(T))\) and \(r_i := c_{R}^{-1}(r(T))\) 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 \((123, 1-)\)-avoiding.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{binary_trees_bitstrings.png}
\caption{Bijections between binary trees and bitstrings.}
\end{figure}
Figure 18. Bijections between pattern-avoiding binary trees and different types of Motzkin paths.
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 $\mathbb{U}$-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.,

$$g(T) := f(L(r_0)), \mathbb{U}, f(L(r_{n-1})), \ldots, \mathbb{U}, f(L(r_1)).$$

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

7.2.3. Bijection between $\mathcal{T}_n(21543, -01-)$ and Motzkin paths with catastrophes $\mathcal{C}_n$. This bijection is illustrated in Figure 18 (c), and it uses as a building block the bijection $g$ defined in the previous section. Instead of $(21543, -01-)$, we consider the mirrored tree pattern $(P,e) := \mu(21543, -01-) = (41235, 01--)$ 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 $(4123, 01-)$. 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 $\mathbb{F}$, $\mathbb{D}$ or $\mathbb{D}_h$ for $h \geq 2$ we obtain a Motzkin path $g'(T_i)$. Note that the rightmost leaf of $T_i$ has no left child, and thus the definition 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 $\mathcal{C}_n$.

7.2.4. Binary trees and Motzkin paths with 2-colored $F$-steps. We now consider Motzkin paths whose $F$-steps come in two possible colors, which we denote by $\mathbb{L}$ and $\mathbb{R}$, respectively. We write $\mathcal{M}'_n$ for the set of $n$-step Motzkin paths with 2-colored $F$-steps. For strings $\tau_1, \ldots, \tau_\ell$, each using symbols from $\{\mathbb{U}, \mathbb{D}, \mathbb{L}, \mathbb{R}\}$, we write $\mathcal{M}'_n(\tau_1, \ldots, \tau_\ell)$ for the Motzkin paths from $\mathcal{M}'_n$ that avoid each of $\tau_1, \ldots, \tau_\ell$ as (consecutive) substrings.

There is a natural bijection $f: \mathcal{T}_n \rightarrow \mathcal{M}'_n$, illustrated in Figure 19. In particular, Motzkin paths with 2-colored $F$-steps are a Catalan family. Given a tree $T \in \mathcal{T}_n$, we define $f(T)$ by considering four cases: If $r(T)$ has no children, then $f(T) := \varepsilon$. If $r(T)$ has only a left child,
then \( f(T) := L, f(L(T)) \). If \( r(T) \) has only a right child, then \( f(T) := R, f(R(T)) \). If \( r(T) \) has two children, then \( f(T) := U, f(R(T)), D, f(L(T)) \).

In the following, we consider the restriction of \( f \) to various sets of binary trees given by pattern avoidance.

**Theorem 12.** The following four mappings \( f : \mathcal{T}_n(P,e) \rightarrow \mathcal{M}'_n(X) \) are bijections:

(i) \((P,e) = (2134,111)\) with \( X = \{U\} \times \{R\} = \{UU,UR\} \);

(ii) \((P,e) = (3214,111)\) with \( X = \{D\} \times \{L\} = \{DU,DL\} \);

(iii) \((P,e) = (1324,111)\) with \( X = \{U,R\} \times \{U\} = \{UU,RU\} \);

(iv) \((P,e) = (1243,111)\) with \( X = \{U,R\} \times \{U,R,L\} = \{UUU,ULL,URU,URL,RRU,RUL,RRU,RRL\} \).

Furthermore, all four sets of Motzkin paths are Wilf-equivalent and counted by OEIS A025242.

**Proof.** The first part of the lemma follows directly from the definition of \( f \). Furthermore, the tree patterns in (i) and (ii) are Wilf-equivalent as they are mirror images of each other. Lastly, the tree patterns in (i), (iii) and (iv) are Wilf-equivalent by Lemma 18  

**Theorem 13.** The following three mappings \( f : \mathcal{T}_n(P,e) \rightarrow \mathcal{M}'_n(X) \) are bijections:

(i) \((P,e) = (21435,1111)\) with \( X = \{UU\} \);

(ii) \((P,e) = (42135,1111)\) with \( X = \{DU\} \);

(iii) \((P,e) = (13254,1111)\) with \( X = \{U,R\} \times \{U,L\} = \{UUU,UUL,RRU,RUL\} \).

Furthermore, all three sets of Motzkin paths are Wilf-equivalent and counted by OEIS A159771.

**Proof.** The first part of the lemma follows directly from the definition of \( f \). Furthermore, the tree patterns in (i) and (ii) are Wilf-equivalent as they are mirror images of each other. Lastly, the tree patterns in (i) and (iii) are Wilf-equivalent by Lemma 22  

7.3. Binary trees and set partitions. In this section, we present bijections between pattern-avoiding binary trees and different pattern-avoiding set partitions.

A **set partition of** \([n]\) is a collection of non-empty disjoint subsets \(B_1,\ldots,B_\ell\), called **blocks**, whose union is \([n]\). A **crossing** in a set partition is a quadruple of elements \(a < b < c < d\) such that \(a,c \in B_i\) and \(b,d \in B_j\) with \(i \neq j\). A partition is called **non-crossing** if it has no crossings. Set partitions are counted by the Bell numbers, and non-crossing set partitions are a well-known Catalan family.

A set partition can be identified uniquely by its **restricted growth string (RGS)**, which is the string \(x_1 \cdots x_n\) given by sorting the blocks by their smallest element, and such that \(x_i = j\) if the element \(i\) is contained in the \(j\)th block in this ordering. For example, the RGS for the partition \([\{1,3\},\{2,4,7\},\{5\},\{6,8\}]\) of \(9\) is 12132424. Restricted growth strings are characterized by the conditions \(x_1 = 1\), and \(x_{i+1} \leq \max\{x_1,x_2,\ldots,x_i\} + 1\) for all \(i = 1,\ldots,n-1\). We write \(P_n\) for the set of all restricted growth strings of set partitions of \([n]\). The notion of pattern containment in permutations extends straightforwardly to pattern containment in strings, in particular in restricted growth strings. Such a pattern string \(\tau\) may contain repeated entries, which means that the corresponding entries in the string in an occurrence of the pattern have the same value. We write \(P_n(\tau_1,\ldots,\tau_\ell)\) for restricted growth strings from \(P_n\) that avoid each the patterns \(\tau_1,\ldots,\tau_\ell\). Observe that \(P_n(1212)\) are precisely non-crossing set partitions. The study of pattern avoidance in set partitions was initiated by Kreweras [Kre72] and Klazar [Kla90, Kla00a, Kla00b]; see also [Goy08, JM08, Sag10, MS11a, MS11b, MS11c, MS13, GP12, JMS13, GGHPT14, BS16].
We also define

\[ \varphi_R(T) = [n]/\sim_R, \]

where \( i \sim_R j \) if \( i \) and \( j \) are in the same right branch of \( T \).

**Lemma 14.** The mappings \( \varphi_L, \varphi_R : \mathcal{T}_n \to \mathcal{P}_n(1212) \) defined in (5) are bijections.

**Proof.** It suffices to prove the statement for \( \varphi_L \), as \( \varphi_R \) is defined symmetrically.

Given \( T \in \mathcal{T}_n \), we first show that the RGS \( \varphi_L(T) \) avoids 1212. Suppose for the sake of contradiction that \( \varphi_L(T) \) contains the pattern 1212, and let \( a < b < c < d \) be the positions of the occurrence of this pattern. This means that in \( T \) the vertices \( a \) and \( c \) are in the same left branch, and the vertices \( b \) and \( d \) are in the same left branch, different from the first one. As \( a \) and \( c \) are in the same branch and \( a < c \) we conclude that \( a \in L(c) \). Furthermore, as \( a < b < c \) we have \( b \in L(c) \). As \( b \) and \( d \) are in the same branch, it follows that \( d \in L(c) \). However, this is a contradiction to \( c < d \).

It is easy to see that the mapping \( \varphi_L \) is injective. To see that it is surjective, consider an RGS \( x \in \mathcal{P}_n(1212) \). The first entry of \( x \) is 1. Let \( j \) be the position of the last 1 in \( x \), and let \( x_L \) and \( x_R \) be the substrings of \( x \) strictly to the left and right of position \( j \), respectively. As \( x \) avoids 1212, no symbol in \( x \) appears both to the left and right of position \( j \). Furthermore, the condition \( x_{i+1} \leq \max\{x_1, x_2, \ldots, x_i\} + 1 \) for all \( i = 1, \ldots, n-1 \) implies that all entries in \( x \) to the left of position \( j \) are strictly smaller than all entries to the right of position \( j \). Consequently, the binary tree \( \varphi_L^{-1}(x) \) has the root \( j \) with the left subtree \( \varphi^{-1}(x_L) \) and the right subtree \( \varphi_L^{-1}(x_R) \).

7.3.2. **Staggered tree patterns.** In the following, we consider the restriction of \( \varphi_L \) and \( \varphi_R \) to various sets of binary trees given by pattern avoidance, and we derive conditions to ensure that
avoiding a tree pattern \( P \) corresponds to avoiding the RGS pattern \( \varphi_L(P) \) or \( \varphi_R(P) \), respectively (in addition to 1212).

A tree pattern \((P, e)\) is called ***staggered***, if one of the following two recursive conditions is satisfied; see Figure 21:

- \((P, e)\) is a contiguous left path, i.e., \( e(i) = 1 \) for all edges \((i, p(i))\) on this path.
- The root of \((P, e)\) is contained in a contiguous left branch, exactly one vertex \( i \) on the branch has a right child \( j := c_R(i) \) with \( e(j) = 0 \), and \((P(j), e_{P(j)})\) is staggered.

\[
(P, e) \quad (\alpha_1, \ldots, \alpha_s; \beta_1, \ldots, \beta_{s-1}) = (6, 4, 5, 2, 3, 2; 5, 2, 5, 1, 2)
\]

\[
\varphi_L(P) = 1111122333334556654221
\]

\[
\begin{array}{c}
\alpha_1 \\
\beta_1 \\
\alpha_2 \\
\beta_2 \\
\alpha_3 \\
\beta_3
\end{array}
\]

\[
\begin{array}{c}
i
\beta_\ell \\
\alpha_\ell
\end{array}
\]

\[
\begin{array}{c}
j = c_R(i)
\end{array}
\]

\[
(P(j), e_{P(j)})
\]

**Figure 21. Definition of staggered tree patterns.**

For the following discussion, for any string \( x \) and any integer \( k \geq 0 \), we write \( x^k \) for the concatenation of \( k \) copies of \( x \). A staggered tree pattern \((P, e)\) is described uniquely by an integer sequence

\[
(\alpha_1, \ldots, \alpha_{s-1}, \alpha_s; \beta_1, \ldots, \beta_{\ell-1})
\]

referred to as its **signature**, where \( \alpha_i \) is the number of vertices on the \( i \)th contiguous left branch, counted from the root, and the \( \beta_\ell \)th vertex counted from bottom to top on the \( i \)th contiguous branch is the unique vertex on that branch having a right child. Clearly, we have \( 1 \leq \beta_\ell \leq \alpha_i \), and furthermore

\[
\varphi_L(P) = 1^{\beta_1}2^{\beta_2}3^{\beta_3} \cdots (\ell - 1)^{\beta_\ell} \ell^{\alpha_i}(\ell - 1)^{\alpha_{i-1}-\beta_\ell-1} \cdots 3^{\alpha_3-\beta_2}2^{\alpha_2-\beta_1}1^{\alpha_1-\beta_1};
\]

see Figure 21

**Theorem 15.** Let \((P, e), P \in \mathcal{T}_k\), be a staggered tree pattern with signature \((\alpha_1, \ldots, \alpha_\ell; \beta_1, \ldots, \beta_{\ell-1})\), satisfying the following two conditions: (i) if \( \ell > 1 \) and \( \beta_\ell = \alpha_1 \), then \( \alpha_1 \in \{1, 2\} \); (ii) \( \beta_i < \alpha_i \) for all \( i = 2, \ldots, \ell - 1 \). Then the mapping \( \varphi_L: \mathcal{T}_n(P, e) \rightarrow \mathcal{P}_n(1212, \varphi_L(P)) \) is a bijection.

**Proof.** By Lemma 14 \( \varphi_L: \mathcal{T}_n \rightarrow \mathcal{P}_n(1212) \) is a bijection. Consequently, if suffices to show that \( T \in \mathcal{T}_n \) contains the tree pattern \((P, e)\) if and only if \( \varphi_L(T) \in \mathcal{P}_n(1212) \) contains the (RGS) pattern \( \varphi_L(P) \).

Using that all left edges \((i, p(i))\) of \((P, e)\) are contiguous, i.e., they satisfy \( e(i) = 1 \), and the definition of the mapping \( \varphi_L \), we see easily that if \( T \in \mathcal{T}_n \) contains the tree pattern \((P, e)\), then \( \varphi_L(T) \) contains the pattern \( \varphi_L(P) \).
It remains to show that if \( x := \varphi_L(T) \) for \( T \in \mathcal{T}_n \) contains the pattern \( \varphi_L(P) \), then \( T \) contains the tree pattern \((P,e)\). For this we argue by induction on \( \ell \), i.e., on the number of contiguous left branches of the staggered pattern. For this part of the argument, conditions (i) and (ii) in the theorem will become relevant. To settle the induction basis, suppose that \( \ell = 1 \), i.e., \((P,e)\) is a contiguous left path. In this case we have \( \varphi_L(P) = 1^k = 1 \cdots 1 \), i.e., \( x \) contains \( k \) occurrences of the same symbol. The definition of \( x = \varphi_L(T) \) shows that consequently, \( T \) contains a left branch on at least \( k \) vertices, i.e., \( T \) contains \((P,e)\), as claimed. For the induction step suppose that \( \ell > 1 \), i.e., the staggered tree pattern \((P,e)\) has at least two contiguous left branches. Consider the contiguous left branch in \( P \) starting at the root, which has \( \alpha_1 \) vertices, and let \( i \) be the \( \beta_1 \)th vertex on this branch counted from bottom to top, which has a right child \( j := c_R(i) \). As \((P,e)\) satisfies conditions (i) and (ii), these conditions are also satisfied for the smaller staggered pattern \((P(j),e_{P(j)})\) (in fact, condition (i) is satisfied trivially). By [6], the pattern \( \varphi_L(P) \) has \( \alpha_1 \) many 1s, split into two groups of size \( \beta_1 \) and \( \alpha_1 - \beta_1 \) that surround all larger symbols. Let \( i_1 < \cdots < i_{\alpha_1} \) be the positions of the symbols to which those 1s are matched in the occurrence of the pattern \( \varphi_L(P) \) in \( x \). By the definition of \( \varphi_L(T) \), the tree \( T \) contains a left branch that includes the vertices \( i_1, \ldots, i_{\alpha_1} \), successively from bottom to top. Let \( x' \) be the substring of \( x \) strictly between positions \( i_{\beta_1} \) and \( i_{\beta_1 + 1} \) if \( \beta_1 < \alpha_1 \) and strictly after position \( i_{\alpha_1} \) if \( \beta_1 = \alpha_1 \). Using that \( x \) is non-crossing and \( \beta_1 < \alpha_1 \) for \( i = 1, \ldots, \ell - 1 \), we may assume w.l.o.g. that \( x' \) does not contain any occurrences of the symbol at positions \( i_1, \ldots, i_{\alpha_1} \) in \( x \). By induction, we know that \( T' := \varphi_L^{-1}(x') \) contains the staggered tree pattern \((P(j),e_{P(j)})\).

We distinguish two cases, namely \( \beta_1 < \alpha_1 \) and \( \beta_1 = \alpha_1 \).

Case (a): \( \beta_1 < \alpha_1 \). By the definition of \( x' \), all vertices in \( T' \) are sandwiched between \( i_{\beta_1} \) and \( i_{\beta_1 + 1} \), implying that \( T' = R(i_{\beta_1}) \) in \( T \). Using that \( T' \) contains the tree pattern \((P(j),e_{P(j)})\), and the property \( e(j) = 0 \) from the definition of staggered tree patterns, we conclude that \( T \) contains the tree pattern \((P,e)\).

Case (b): \( \beta_1 = \alpha_1 \). By the definition of \( x' \), all vertices in \( T' \) are larger than \( i_{\alpha_1} \). If \( T' = R(i_{\alpha_1}) \) in \( T \), then \( T \) contains the tree pattern \((P,e)\), as argued in the previous case. Otherwise, consider the lowest common ancestor \( \hat{i} \) of \( i_{\alpha_1} \) and \( T' \) in \( T \). Specifically, we have \( i_1, \ldots, i_{\alpha_1} \in L(i) \) and \( T' \subseteq R(i) \) in \( T \). Condition (i) stated in the theorem asserts that \( \alpha_1 \in \{1,2\} \), and therefore \( T \) contains the tree pattern \((P,e)\). Specifically, \( \hat{i} \) and its left child in \( T \) make the occurrence of \((P(j),e_{P(j)})\) to an occurrence of \((P,e)\).

This completes the proof of the theorem.

By applying the mirroring operation \( \mu \) to a tree pattern, we obtain the following immediate consequence of Theorem [15]

**Theorem 16.** Let \((P,e)\) be such that \( \mu(P,e) \) is staggered and satisfies the conditions of Theorem [15]. Then the mapping \( \varphi_R : \mathcal{T}_n(P,e) \to \mathcal{P}_n(1212, \varphi_R(P)) \) is a bijection.

Theorems [15] and [16] are quite versatile. Applying them to all tree patterns on at most 5 vertices, we obtain the correspondences with pattern-avoiding set partitions listed in Table [4]. This also establishes some interesting Wilf-equivalences between various pattern-avoiding non-crossing set partitions, for example between the three sets \( \mathcal{P}_n(1212,1232), \mathcal{P}_n(1212,1221) \), and \( \mathcal{P}_n(1212,1213) \), or between the three sets \( \mathcal{P}_n(1212,12332), \mathcal{P}_n(1212,12213) \), and \( \mathcal{P}_n(1212,12321) \) (cf. [MS11b]).

8. Wilf-equivalence of tree patterns

In this section we provide five general lemmas for establishing Wilf-equivalence of certain tree patterns that are obtained by replacing some subpattern \((P,e)\) with a Wilf-equivalent
Table 4. All tree patterns $(P, e)$ on at most 5 vertices for which $(P, e)$ or $\mu(P, e)$ is staggered and the corresponding RGS patterns given by Theorems 15 or 16 respectively.

<table>
<thead>
<tr>
<th>Tree Patterns</th>
<th>RGS patterns</th>
<th>Bijection</th>
<th>OEIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>(123, 11) = (123, 1-)</td>
<td>1212, 111</td>
<td>$\varphi_R$</td>
<td>A001006</td>
</tr>
<tr>
<td>(132, 10) = (132, -)</td>
<td>1212, 121</td>
<td>$\varphi_R$</td>
<td>A000079</td>
</tr>
<tr>
<td>(213, 10) = (213, -)</td>
<td>1212, 112</td>
<td>$\varphi_L$</td>
<td>A000079</td>
</tr>
<tr>
<td>(213, 01) = (213, -)</td>
<td>1212, 122</td>
<td>$\varphi_R$</td>
<td>A000079</td>
</tr>
<tr>
<td>(1234, 111) = (1234, 11-)</td>
<td>1212, 1111</td>
<td>$\varphi_R$</td>
<td>A036765</td>
</tr>
<tr>
<td>(1234, 110) = (1234, 1-)</td>
<td>1212, 1112</td>
<td>$\varphi_R$</td>
<td>A025242</td>
</tr>
<tr>
<td>(4312, 110) = (4312, 1-)</td>
<td>1212, 1211</td>
<td>$\varphi_R$</td>
<td>A025242</td>
</tr>
<tr>
<td>(1324, 101) = (1324, 1-)</td>
<td>1212, 1211</td>
<td>$\varphi_R$</td>
<td>A025242</td>
</tr>
<tr>
<td>(4213, 110) = (4213, 1-)</td>
<td>1212, 1211</td>
<td>$\varphi_R$</td>
<td>A025242</td>
</tr>
<tr>
<td>(1423, 010) = (1423, 0-)</td>
<td>1212, 1232</td>
<td>$\varphi_L$</td>
<td>A001519</td>
</tr>
<tr>
<td>(1423, 101) = (1423, -0-)</td>
<td>1212, 1221</td>
<td>$\varphi_R$</td>
<td>A001519</td>
</tr>
<tr>
<td>(4132, 010) = (4132, 0-)</td>
<td>1212, 1213</td>
<td>$\varphi_R$</td>
<td>A001519</td>
</tr>
<tr>
<td>(1432, 011) = (1432, -1-)</td>
<td>1212, 1222</td>
<td>$\varphi_L$</td>
<td>A005773</td>
</tr>
<tr>
<td>(4123, 011) = (4123, -1-)</td>
<td>1212, 1112</td>
<td>$\varphi_R$</td>
<td>A005773</td>
</tr>
<tr>
<td>(2143, 101) = (2143, -0-)</td>
<td>1212, 1122</td>
<td>$\varphi_L$</td>
<td>A001519</td>
</tr>
<tr>
<td>(12345, 1111) = (12345, 111-)</td>
<td>1212, 11111</td>
<td>$\varphi_R$</td>
<td>A036766</td>
</tr>
<tr>
<td>(12345, 1110) = (12345, 11-)</td>
<td>1212, 11121</td>
<td>$\varphi_R$</td>
<td>A159768</td>
</tr>
<tr>
<td>(4312, 1110) = (4312, 11-)</td>
<td>1212, 12111</td>
<td>$\varphi_R$</td>
<td>A159768</td>
</tr>
<tr>
<td>(12435, 1101) = (12435, 11-)</td>
<td>1212, 12111</td>
<td>$\varphi_R$</td>
<td>A159768</td>
</tr>
<tr>
<td>(12534, 1101) = (12534, 10-)</td>
<td>1212, 12211</td>
<td>$\varphi_R$</td>
<td>A176677</td>
</tr>
<tr>
<td>(54132, 1101) = (54132, 10-)</td>
<td>1212, 12211</td>
<td>$\varphi_R$</td>
<td>A176677</td>
</tr>
<tr>
<td>(13245, 1011) = (13245, 1-)</td>
<td>1212, 12111</td>
<td>$\varphi_R$</td>
<td>A159768</td>
</tr>
<tr>
<td>(53214, 1110) = (53214, 11-)</td>
<td>1212, 11121</td>
<td>$\varphi_L$</td>
<td>A159768</td>
</tr>
<tr>
<td>(14235, 1011) = (14235, 10-)</td>
<td>1212, 12211</td>
<td>$\varphi_R$</td>
<td>A176677</td>
</tr>
<tr>
<td>(52143, 1101) = (52143, 1-)</td>
<td>1212, 12211</td>
<td>$\varphi_L$</td>
<td>A176677</td>
</tr>
<tr>
<td>(15234, 1011) = (15234, -01-)</td>
<td>1212, 12221</td>
<td>$\varphi_R$</td>
<td>A054391</td>
</tr>
<tr>
<td>(15243, 0101) = (15243, 0-0-)</td>
<td>1212, 12332</td>
<td>$\varphi_L$</td>
<td>A007051</td>
</tr>
<tr>
<td>(51423, 0101) = (51423, 00-)</td>
<td>1212, 12213</td>
<td>$\varphi_R$</td>
<td>A007051</td>
</tr>
<tr>
<td>(15243, 1010) = (15243, -0-)</td>
<td>1212, 12321</td>
<td>$\varphi_R$</td>
<td>A007051</td>
</tr>
<tr>
<td>(15324, 0101) = (15324, 01-)</td>
<td>1212, 12232</td>
<td>$\varphi_L$</td>
<td>A365508</td>
</tr>
<tr>
<td>(51324, 0101) = (51324, 01-)</td>
<td>1212, 12113</td>
<td>$\varphi_R$</td>
<td>A365508</td>
</tr>
<tr>
<td>(15423, 0101) = (15423, 01-)</td>
<td>1212, 12322</td>
<td>$\varphi_L$</td>
<td>A365508</td>
</tr>
<tr>
<td>(51423, 0101) = (51423, 01-)</td>
<td>1212, 12113</td>
<td>$\varphi_R$</td>
<td>A365508</td>
</tr>
<tr>
<td>(15432, 0111) = (15432, -11-)</td>
<td>1212, 12222</td>
<td>$\varphi_L$</td>
<td>A159772</td>
</tr>
<tr>
<td>(51324, 0111) = (51324, -11-)</td>
<td>1212, 11122</td>
<td>$\varphi_R$</td>
<td>A159772</td>
</tr>
<tr>
<td>(21534, 1010) = (21534, -0-)</td>
<td>1212, 12322</td>
<td>$\varphi_L$</td>
<td>A007051</td>
</tr>
<tr>
<td>(41325, 0101) = (41325, 00-)</td>
<td>1212, 12133</td>
<td>$\varphi_R$</td>
<td>A007051</td>
</tr>
<tr>
<td>(21543, 1011) = (21543, -01-)</td>
<td>1212, 12222</td>
<td>$\varphi_L$</td>
<td>A054391</td>
</tr>
<tr>
<td>(41235, 0111) = (41235, 01-)</td>
<td>1212, 11122</td>
<td>$\varphi_R$</td>
<td>A054391</td>
</tr>
</tbody>
</table>
subpattern \((P', e')\), or by moving it to a different vertex in the surrounding tree pattern. We also give results for two specific patterns on 5 vertices: a Wilf-equivalence that is not covered by the general lemmas and a counting argument based on a Catalan-like recurrence. We apply these results to systematically study Wilf-equivalences between all tree patterns on at most 5 vertices; see Tables 1–3.

8.1. Subpattern replacement and shifting lemmas. The first lemma considers replacing a subpattern with a Wilf-equivalent subpattern attached by a non-contiguous edge to a contiguous tree; see Figure 22.

**Lemma 17.** Let \((S, 1 \cdots 1)\) be a contiguous tree pattern, and let \(x\) be a vertex in \(S\) that does not have a right child. Let \(Q\) and \(Q'\) denote the tree patterns obtained from \((S, 1 \cdots 1)\) by attaching tree patterns \((P, e)\) or \((P', e')\) with \(P, P' \in T_b\), respectively, with a non-contiguous edge to \(x\) as a right subtree. If \((P, e)\) and \((P', e')\) are Wilf-equivalent, then \(Q\) and \(Q'\) are also Wilf-equivalent.

Note that while all edges of \((S, 1 \cdots 1)\) are contiguous, no assumption is made about the edges of \(P\) or \(P'\), i.e., the functions \(e\) and \(e'\) are arbitrary. However, the edge from \(x\) to the root of \((P, e)\) or \((P', e')\) must be non-contiguous in both \(Q\) and \(Q'\).

**Proof.** The proof is illustrated in Figure 22. As \((P, e)\) and \((P', e')\) are Wilf-equivalent, there is a bijection \(g_n : T_n(P, e) \to T_n(P', e')\) for all \(n \geq 0\), and we let \(g\) be the union of those functions over all \(n \geq 0\). Let \(T \in T_n(Q)\) and consider all occurrences of \((S, 1 \cdots 1)\) in \(T\). Let \(X = \{x_1, \ldots, x_\ell\}\) be the corresponding occurrences of the vertex \(x\) of \(Q\) in the host tree \(T\), i.e., \(x_i \in [n]\) denotes the vertex to which \(x\) is mapped in the \(i\)th occurrence of \((S, 1 \cdots 1)\), for all \(i = 1, \ldots, \ell\). Furthermore, let \(X' \subseteq X\) be minimal such that every \(x_i \in X\) has some predecessor in \(X'\). In other words, \(X'\) is the subset of vertices of \(X\) that mark the first occurrences of \((S, 1 \cdots 1)\) when looking from the root of \(T\). As \(T\) avoids \(Q\), the subtree \(R(x_i)\) avoids \((P, e)\) for every \(x_i \in X\). We define \(f(T) \in T_n\) as the tree obtained from \(T\) by replacing \(R(x_i)\) with \(g(R(x_i))\) for every \(x_i \in X'\). Although this may introduce new occurrences of \(S\) in \(f(T)\), these new occurrences have their vertex \(x\) mapped to a vertex inside some subtree \(g(R(x_i))\) with \(x_i \in X'\). Thus, since \(g(R(x_i))\) for each \(x_i \in X'\) avoids \((P', e')\), the tree \(f(T)\) avoids \(Q'\). Furthermore, since the set \(X'\) is invariant under \(f\), the mapping \(f\) is reversible, so it is a bijection. \(\square\)

Our second lemma considers moving a subpattern attached by a non-contiguous edge along a left branch; see Figure 23.

**Lemma 18.** Let \(P_d\) denote the left path on \(d\) vertices and let \(x, x'\) be two distinct vertices of \(P_d\). Let \(Q\) and \(Q'\) denote the tree patterns obtained from the contiguous pattern \((P_d, 1 \cdots 1)\) by attaching a tree pattern \((P, e)\) with a non-contiguous edge to either \(x\) or \(x'\) as a right subtree, respectively. Then \(Q\) and \(Q'\) are Wilf-equivalent.

**Proof.** The proof is illustrated in Figure 23. Let \(s\) and \(s'\) be the number of vertices above and including \(x\) or \(x'\) in \(Q\) and \(Q'\), respectively. We assume w.l.o.g. that \(s < s'\). We inductively describe a bijection \(f : T_n(Q) \to T_n(Q')\). Consider a tree \(T \in T_n(Q)\). If \(n\) is strictly smaller than the number of vertices of \(Q\), then we define \(f(T) := T\), i.e., \(f\) is defined to be the identity mapping. Otherwise we define \(b := \beta_L(r(T))\) and \(\ell_i := c_L^{-1}(r(T))\) for \(i = 1, \ldots, b\), i.e., we consider the left branch \((\ell_1, \ldots, \ell_b)\) starting at the root of \(T\). If \(b < d\) then \(f(T)\) is obtained by recursively applying \(f\) to each subtree \(R(\ell_i)\) for \(i = 1, \ldots, b\). It remains to consider the case that \(b \geq d\). Since \(T\) avoids \(Q\), each subtree \(R(\ell_i)\) for \(i = s, \ldots, b - (d - s)\) avoids \((P, e)\) and each subtree \(R(\ell_i)\) for \(i = 1, \ldots, s - 1\) and \(i = b - (d - s - 1), \ldots, b\) avoids \(Q\). The tree \(f(T)\) is obtained from \(T\) by cyclically down-shifting the left branch \((\ell_1, \ldots, \ell_b)\) and its right subtrees by...
$T \in \mathcal{T}_n(Q)$ \hspace{1cm} $f(T) \in \mathcal{T}_n(Q')$

\[ X = \{x_1, x_2, x_3, x_4\} \] \hspace{1cm} \[ X' = \{x_1, x_2\} \]

Figure 22. Illustration of Lemma 19. The bijection $g: \mathcal{T}_n(213, \cdots) \rightarrow \mathcal{T}_n(132, \cdots)$ is the composition of the bijections described in Sections 7.1.3 and 7.1.1.

The next four lemmas consider different ways of moving a subpattern attached by a non-contiguous edge to a contiguous L-shaped path of length 2; see Figures 24–26.

**Lemma 19.** Let $S$ and $S'$ be the paths of length 2 with $\tau(S) = 132$ and $\tau(S') = 213$, respectively. Let $Q$ and $Q'$ denote the tree patterns obtained from the contiguous patterns $(S, 11)$ and $(S', 11)$ by attaching a tree pattern $(P, e)$ with a non-contiguous edge to the root of $S$ as a left subtree and to the right leaf of $S'$ as a left subtree, respectively. Then $Q$ and $Q'$ are Wilf-equivalent.

**Proof.** The proof is illustrated in Figure 24. We inductively describe a bijection $f: \mathcal{T}_n(Q) \rightarrow \mathcal{T}_n(Q')$. Consider a tree $T \in \mathcal{T}_n(Q)$. If $n$ is strictly smaller than the number of vertices of $Q$, then we define $f(T) := T$, i.e., $f$ is defined to be the identity mapping. Otherwise 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 at the root of $T$. Since $T$ avoids $Q$, in every maximal sequence of consecutive non-empty left subtrees $L(r_i) \neq \varepsilon$, the last one avoids $Q$ and all earlier ones avoid $(P, e)$. The tree $f(T)$ is obtained from $T$ by reversing the order of vertices and their left subtrees on the branch $(r_1, \ldots, r_b)$, and by applying $f$ recursively to the subtrees avoiding $Q$ (i.e., the last subtree in each maximal sequence of consecutive non-empty left subtrees $L(r_i) \neq \varepsilon$ of $T$). By construction, the tree $f(T)$
avoids the tree pattern $Q'$. Furthermore, the mapping $f: \mathcal{T}_n(Q) \rightarrow \mathcal{T}_n(Q')$ is reversible, so it is a bijection. 

**Lemma 20.** Let $S$ and $S'$ be the paths of length 2 with $\tau(S) = 132$ and $\tau(S') = 213$, respectively. Let $Q$ and $Q'$ denote the tree patterns obtained from the contiguous patterns $(S, 11)$ and $(S', 11)$ by attaching a tree pattern $(P, e)$ with a non-contiguous edge to the leaf of $S$ as a right subtree and to the left leaf of $S'$ as a left subtree, respectively. Then $Q$ and $Q'$ are Wilf-equivalent.

**Proof.** The proof is illustrated in Figure 25. We inductively describe a bijection $f: \mathcal{T}_n(Q) \rightarrow \mathcal{T}_n(Q')$. Consider a tree $T \in \mathcal{T}_n(Q)$. If $n$ is strictly smaller than the number of vertices of $Q$, then we define $f(T) := T$, i.e., $f$ is defined to be the identity mapping. Otherwise we define $b := \beta_R(r(T))$ and $r_i := \ell_{i-1}^R(r(T))$ for $i = 1, \ldots, b$, i.e., we consider the right branch $(r_1, \ldots, r_b)$ starting at the root of $T$. Since $T$ avoids $Q$, the subtree $L(r_1)$ avoids $Q$ and for each $i = 2, \ldots, b$, if $c_L(r_i) \neq \varepsilon$ then $L(c_L(r_i))$ avoids $Q$ and $R(c_L(r_i))$ avoids $(P, e)$. The tree $f(T)$ is obtained from $T$ by reversing the order of vertices and their left subtrees on the branch $(r_1, \ldots, r_b)$, by swapping the subtrees $L(c_L(r_i))$ and $R(c_L(r_i))$ if $c_L(r_i) \neq \varepsilon$ for $i = 2, \ldots, b$, and by applying $f$ recursively to the subtrees avoiding $Q$ (i.e., the subtree $L(r_1)$ of $T$ and the subtrees $L(c_L(r_i))$ if $c_L(r_i) \neq \varepsilon$ for $i = 2, \ldots, b$). By construction, the tree $f(T)$ avoids the tree pattern $Q'$. Furthermore, the mapping $f: \mathcal{T}_n(Q) \rightarrow \mathcal{T}_n(Q')$ is reversible, so it is a bijection. 

**Lemma 21.** Let $S$ be the path of length 2 with $\tau(S) = 132$. Let $Q$ and $Q'$ denote the tree patterns obtained from the contiguous pattern $(S, 11)$ by attaching a tree pattern $(P, e)$ with a
non-contiguous edge to either the leaf or the middle vertex of $S$ as a right subtree, respectively. Then $Q$ and $Q'$ are Wilf-equivalent.

Proof. The proof is illustrated in Figure 26. We inductively describe a bijection $f : \mathcal{T}_n(Q) \to \mathcal{T}_n(Q')$. Consider a tree $T \in \mathcal{T}_n(Q)$. If $n$ is strictly smaller than the number of vertices of $Q$, then we define $f(T) := T$, i.e., $f$ is defined to be the identity mapping. Otherwise we define $b := \beta_R(r(T))$ and $r_i := c_i^R(r(T))$ for $i = 1, \ldots, b$, i.e., we consider the right branch $(r_1, \ldots, r_b)$ starting at the root of $T$. Since $T$ avoids $Q$, the subtree $L(r_1)$ avoids $Q$ and for each $i = 2, \ldots, b$, if $c_L(r_i) \neq \varepsilon$ then $L(c_L(r_i))$ avoids $Q$ and $R(c_L(r_i))$ avoids $(P,e)$. The tree $f(T)$ is obtained from $T$ as follows. First, we replace $L(r_1)$ recursively by $f(L(r_1))$. Next, we consider every $i \in \{2, \ldots, b\}$ for which $c_L(r_i) \neq \varepsilon$, we replace $L(c_L(r_i))$ recursively by $f(L(c_L(r_i)))$, and we swap the subtrees $R(c_L(r_i))$ and $R(r_i)$, i.e., $R(c_L(r_i))$ is attached as the right subtree of $r_i$, and $R(r_i)$ is attached as the right subtree of $c_L(r_i)$. Some of these subtrees may be empty $\varepsilon$, then by attaching an empty tree we mean attaching no tree. In particular, $R(r_b)$ is empty. By construction, the tree $f(T)$ avoids the tree pattern $Q'$. Furthermore, the mapping $f : \mathcal{T}_n(Q) \to \mathcal{T}_n(Q')$ is reversible, as the zigzag path on the vertices $r_i$ and $c_L(r_i)$ if they exist, for $i = 2, \ldots, b$, is uniquely determined in $f(T)$. Consequently, $f$ is a bijection, as claimed. \hfill \Box

Clearly, by applying the mirroring operation $\mu$, we obtain variants of the preceding lemmas where the direction of attachment is interchanged.

8.2. Specific patterns with 5 vertices.
Lemma 22. The tree patterns $\mathcal{T}_n(21435, -1-\cdot-)$ and $\mathcal{T}_n(13254, 1-1-\cdot)$ are Wilf-equivalent.

The proof uses the path reversal bijection technique employed in the proofs of Lemmas 19 and 20; see Figure 27. We omit the details.

Lemma 23. The class of trees $\mathcal{T}_n(12435, 10-\cdot)$ is counted by the sequence OEIS A176677.

Proof. The sequence $(a_n)_{n\geq 1}$ in OEIS A176677 is defined by the recurrence $a_0 = a_1 = 1$ and $a_{n+1} = -1 + \sum_{p=0}^{n-1} a_pa_{n-p}$ for $n \geq 1$. Using this definition, a straightforward computation shows that $t_n := a_{n+1}$ satisfies the recursion $t_0 = t_1 = 1$ and

$$t_{n+1} = 2t_n - 1 + \sum_{p=0}^{n-1} t_pt_{n-1-p} \quad \text{for } n \geq 1. \quad (7)$$

To prove the lemma, we show that $t_n = |\mathcal{T}_n(P, e)|$ for $(P, e) := (12435, 10-\cdot)$ by induction.

Clearly, we have $t_0 = 1$ as $\mathcal{T}_0(P, e) = \{\varepsilon\}$, which settles the induction basis. For the induction step, let $n \geq 1$ and consider all trees $T$ from $\mathcal{T}_{n+1}(P, e)$, distinguished by the number $b := \beta_T(r(T))$ of vertices on the right branch $(r_1, \ldots, r_b)$ starting at the root. Clearly, there are exactly $t_n$ trees $T$ with $b = 1$ as $L(T) \in \mathcal{T}_n(P, e)$ and $R(T) = \varepsilon$ in such trees. Furthermore, the number of trees $T$ with $b = 2$ is $\sum_{p=0}^{n-1} t_pt_{n-1-p}$ as $L(T) = L(r_1) \in \mathcal{T}_p(P, e)$ and $L(r_2) \in \mathcal{T}_{n-1-p}(P, e)$ in such trees where $p = |L(T)|$ ranges from 0 to $n - 1$.

We claim that the number of trees with $b \geq 3$ is $t_n - 1$ by mapping them bijectively to all trees in $\mathcal{T}_n(P, e)$ except the left path. Let $T \in \mathcal{T}_{n+1}(P, e)$ with $b \geq 3$. Observe that $T(r_3)$ is a zigzag path as $T$ avoids $(P, e)$. Let $c := \beta_L(r_3)$ and let $(r_3 = \ell_1, \ldots, \ell_c)$ denote the left branch in $T$ starting at $r_3$. We map $T \in \mathcal{T}_{n+1}(P, e)$ to a tree $f(T) \in \mathcal{T}_n(P, e)$ as follows; see Figure 28.
$T \in T_n(Q)$

$T \in T_{n}(Q)$

$f(T) \in T_{n}(Q')$

$f(T) \in T_{n}(Q')$

$f(L(r_1))$ avoids $Q$

$f(L(r_1))$ avoids $Q$

$R(c_L(r_i))$ avoids $(P, e)$

$R(c_L(r_i))$ avoids $(P, e)$

Figure 26. Illustration of Lemma 21.

$(P, e) = (21435, 1111)$

$(P', e') = (13254, 1111)$

Figure 27. Illustration of Lemma 22.
We remove the branch \((\ell_1, \ldots, \ell_c)\) from \(T\), make \(\ell_1\) the root of \(f(T)\), identify \(\ell_c\) with \(r_1\) (so these two vertices merge into one), and the subtree \(R(\ell_c)\) (possibly empty) is attached to \(r_2\) as a right child instead of \(r_3 = \ell_1\). Observe that \(f(T) \in T_n(P, e)\), the tree \(f(T)\) is not the left path, and the mapping \(f\) is reversible. This completes the inductive proof of (7). \(\square\)

8.3. Wilf-equivalent patterns with up to 5 vertices. In this section we apply the lemmas derived in the preceding sections to establish Wilf-equivalences between tree patterns on at most 5 vertices; see Figures 29–31. If needed, we use mirrored variants of the lemmas, which is not shown in the figures.

It remains an open problem to find a bijection between the tree patterns \((31245, 0-1-)\) and \((31245, 1-0-)\), or between any of their Wilf-equivalent patterns. The first class of trees is counted
by OEIS A176677, as it is Wilf-equivalent to (12435, 10--), and then we can use Lemma 23. For the second class of trees we are missing an argument connecting it to the first class.

9. 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 the first family of objects, such a bijection might help to prove Wilf-equivalence between the tree patterns (31245, 0-1-) and (31245, 1-0-), or between any of their Wilf-equivalent patterns.

- For purely contiguous or non-contiguous tree patterns \((P, e)\), there are recursions to derive the generating function for \(|T_n(P, e)|\); see [Row10] and [DPTW12]. For our more general patterns with some contiguous and some non-contiguous edges, these methods seem to fail. Therefore, it is an interesting open question whether there is an algorithm to compute those more general generating functions, and to understand some of their properties. Furthermore, can the set of pattern-avoiding trees for such pure (non-friendly) patterns be generated efficiently?
Figure 31. Wilf classes of tree patterns on 5 vertices (see legend in Figure 29; the question mark denotes an open problem). Non-contiguous patterns are omitted as they are all counted by OEIS A007051 ([DPTW12, Thm. 1]). Also contiguous patterns where no lemma applies are omitted. Only two out of 13 patterns from the class NewA are shown.
• 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 \(f(i) = c'_l(f(p(i)))\) for some \(j > 0\), whereas if \(i = c_R(p(i))\), then \(f(i) = c'_R(f(p(i)))\) for some \(j > 0\). Theorem 2 can be generalized to also capture this new notion, by modifying the definition \([25]\) 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 3.2 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 \texttt{cos}.

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 [MM23].

**References**


