PATTERN AVOIDANCE IN BINARY TREES

ERIC S. ROWLAND

Abstract. This paper considers the enumeration of trees avoiding a contiguous pattern. We provide an algorithm for computing the generating function that counts n-leaf binary trees avoiding a given binary tree pattern t. Equipped with this counting mechanism, we study the analogue of Wilf equivalence in which two tree patterns are equivalent if the respective n-leaf trees that avoid them are equinumerous. We investigate the equivalence classes combinatorially, finding some relationships to Dyck words avoiding a given subword. Toward establishing bijective proofs of tree pattern equivalence, we develop a general method of restructuring trees that conjecturally succeeds to produce an explicit bijection for each pair of equivalent tree patterns.

1. Introduction

Determining the number of words of length n on a given alphabet that avoid a certain (contiguous) subword is a classical combinatorial problem that can be solved, for example, by the principle of inclusion–exclusion. An approach to this question using generating functions is provided by the Goulden–Jackson cluster method [6, 10], which utilizes only the self-overlaps (or "autocorrelations") of the word being considered. A natural question is "When do two words have the same avoiding generating function?" That is, when are the n-letter words avoiding (respectively) w_1 and w_2 equinumerous for all n? The answer is simple: precisely when their self-overlaps coincide. For example, the equivalence classes of length-4 words on the alphabet $\{0, 1\}$ are as follows.

equivalence class	self-overlap lengths
$\{0001, 0011, 0111, 1000, 1100, 1110\}$	$\{4\}$
$\{0010, 0100, 0110, 1001, 1011, 1101\}$	$\{1, 4\}$
$\{0101, 1010\}$	$\{2, 4\}$
$\{0000, 1111\}$	$\{1, 2, 3, 4\}$

In this paper we consider the analogous questions for plane trees. All trees in the paper are rooted and ordered. The *depth* of a vertex is the length of the minimal path to that vertex from the root, and depth(T) is the maximum vertex depth in the tree T.

Our focus will be on *binary trees* — trees in which each vertex has 0 or 2 (ordered) children. A vertex with 0 children is a *leaf*, and a vertex with 2 children is an *internal* vertex. A binary tree with n leaves has $n - 1$ internal vertices, and the number of such trees is the Catalan number C_{n-1}. The first few binary trees are depicted in Figure 1. We use an indexing for n-leaf binary trees that arises from the natural recursive construction of all n-leaf binary trees by pairing each k-leaf binary tree

Date: February 8, 2010.

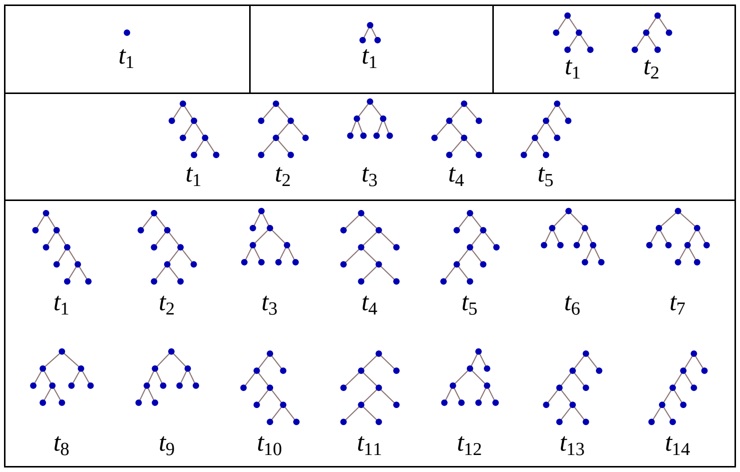

t_1
t_1
t_1 t_2
t_1 t_2 t_3 t_4 t_5
t_1 t_2 t_3 t_4 t_5 t_6 t_7
t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14}

FIGURE 1. The binary trees with at most 5 leaves.

with each $(n-k)$-leaf binary tree, for all $1 \leq k \leq n-1$. In practice it will be clear from context which tree we mean by, for example, 't_1'.

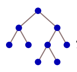
Conceptually, a binary tree T *avoids* a tree pattern t if there is no instance of t anywhere inside T. Steyaert and Flajolet [15] were interested in such patterns in vertex-labeled trees. They were mainly concerned with the asymptotic probability of avoiding a pattern, whereas our focus is on enumeration. However, they establish in Section 2.2 that the total number of occurrences of an m-leaf binary tree pattern t in all n-leaf binary trees is

$$\binom{2n-m}{n-m}.$$

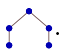
In this sense, all m-leaf binary trees are indistinguishable; the results of this paper refine this statement by further distinguishing m-leaf tree patterns by the number of n-leaf trees containing precisely k copies of each.

We remark that a different notion of tree pattern was later considered by Flajolet, Sipala, and Steyaert [5], in which every leaf of the pattern must be matched by a leaf of the tree. Such patterns are only matched at the bottom of a tree, so they arise naturally in the problem of compactly representing in memory an expression containing repeated subexpressions. The enumeration of trees avoiding such a pattern is simple, since no two instances of the pattern can overlap: The number of n-leaf binary trees avoiding t depends only on the number of leaves in t. See also Flajolet and Sedgewick [4, Note III.40].

The reason for studying patterns in binary trees as opposed to rooted, ordered trees in general is that it is straightforward to determine what it should mean for a binary tree to avoid, for example,

$t_7 =$ ,

whereas *a priori* it is ambiguous to say that a general tree avoids

.

Namely, for general trees, 'matches a vertex with i children' for $i \geq 1$ could mean either 'has exactly i children' or 'has at least i children'. For binary trees, these are the same for $i = 2$, so there is no choice to be made.

However, it turns out that the notion of pattern avoidance for binary trees induces a well-defined notion of pattern avoidance for general trees. This arises via the natural bijection β between the set of n-leaf binary trees and the set of n-vertex trees; using this bijection, one simply translates the problem into the setting of binary trees.

One main theoretical purpose of this paper is to provide an algorithm for computing the generating function that counts binary trees avoiding a certain tree pattern. This algorithm easily generalizes to count trees containing a prescribed number of occurrences of a certain pattern, and additionally we consider the number of trees containing several patterns each a prescribed number of times. All of these generating functions are algebraic. Section 4 is devoted to these algorithms, which are implemented in TreePatterns [11], a *Mathematica* package available from the author's website.

By contrast, another main purpose of this paper is quite concrete, and that is to determine equivalence classes of binary trees. We say that two tree patterns s and t are *equivalent* if for all $n \geq 1$ the number of n-leaf binary trees avoiding s is equal to the number of n-leaf binary trees avoiding t. In other words, equivalent trees have the same generating function with respect to avoidance. This is the analogue of Wilf equivalence in permutation patterns. Each tree is trivially equivalent to its left–right reflection, but there are other equivalences as well. The first few classes are presented in Section 3. The appendix contains a complete list of equivalence classes of binary trees with at most 6 leaves, from which we draw examples throughout the paper. Classes are named with the convention that class $m.i$ is the ith class of m-leaf binary trees.

We seek to understand equivalence classes of binary trees combinatorially, and this is the third purpose of the paper. By analogy with words, one might hope for a simple criterion such as "s and t are equivalent precisely when the lengths of their self-overlaps coincide"; however, although the set of self-overlap lengths seems to be preserved under equivalence, this statement is not true, for $\{1, 1, 2, 2, 5\}$ corresponds to both classes 6.3 and 6.7. In lieu of a simple criterion, we look for bijections. As discussed in Section 3.5, in a few cases there is a bijection between n-leaf binary trees avoiding a certain pattern and Dyck $(n-1)$-words avoiding a certain (contiguous) subword. In general, when s and t are equivalent tree patterns, we would like to provide a bijection between trees avoiding s and trees avoiding t. Conjecturally, all classes of binary trees can be established bijectively by *top-down* and *bottom-up* replacements; this is the topic of Section 5. Nearly all bijections in the paper are implemented in the package TreePatterns.

Aside from mathematical interest, a general study of pattern avoidance in trees has applications to any collection of objects related by a tree structure, such as people in a family tree or species in a phylogenetic tree. In particular, this paper answers the following question. Given n related objects (e.g., species) for which the exact relationships aren't known, how likely is it that some prescribed (e.g., evolutionary) relationship exists between some subset of them? (Unfortunately, it probably will not lead to insight regarding the practical question "What is the probability of avoiding a mother-in-law?") Alternatively, we can think of trees

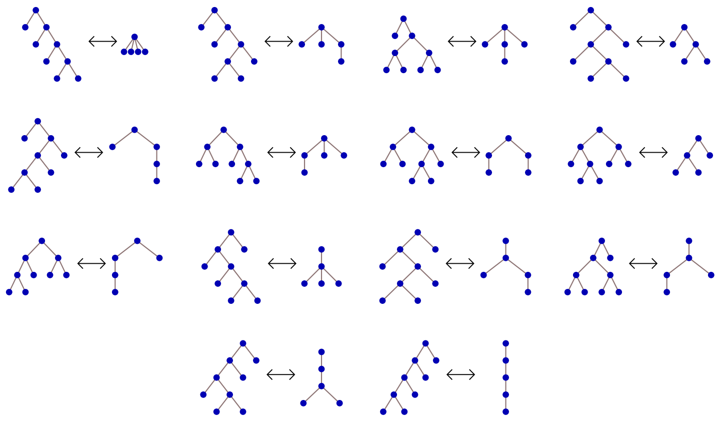


FIGURE 2. The Harary–Prins–Tutte correspondence β between binary trees with 5 leaves and trees with 5 vertices.

as describing the syntax of sentences in natural language or of fragments of computer code; in this context the paper answers questions about the occurrence and frequency of given phrase substructures.

2. Definitions

2.1. **The Harary–Prins–Tutte bijection.**

We first recall a fundamental bijection between n-leaf binary trees and general (rooted, ordered) n-vertex trees. The bijection was given by Harary, Prins, and Tutte [7] and simplified by de Bruijn and Morselt [1]. Following Knuth [9, Section 2.3.2], we use a modified version in which the trees are de-planted. (An extra vertex is used by those authors because they think of these objects as trivalent trees.) The correspondence for $n = 5$ is shown in Figure 2. Throughout the paper we shall call this bijection β. That is,

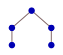
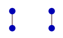
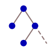
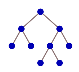
$$\beta : (\text{set of binary trees}) \to (\text{set of all trees}).$$

To obtain the n-vertex tree $\beta(T)$ associated with a given n-leaf binary tree T, contract every rightward edge to a single vertex.

The inverse map is a little more tedious to describe; however, one simply reverses the algorithm. To obtain the n-leaf binary tree $\beta^{-1}(T)$ associated with a given n-vertex tree T:

(1) Delete the root vertex.
(2) For each remaining vertex, let its new left child be its original leftmost child (if it exists), and let its new right child be its original immediate right sibling (if it exists).
(3) Add children to the existing vertices so that each has two children. (If a vertex has only one child, the new child is added in place of the child missing in step 2.) Note that the leaves of the final binary tree are precisely the vertices added in this step.

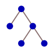
For example,

$T =$  $\xrightarrow{(1)}$  $\xrightarrow{(2)}$  $\xrightarrow{(3)}$  $= \beta^{-1}(T)$.

If T is an n-vertex tree, then clearly $\beta^{-1}(T)$ is a binary tree, and $\beta^{-1}(T)$ has n leaves because the $n-1$ vertices present in step 2 are precisely the internal vertices of $\beta^{-1}(T)$.

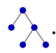
Of course, β is chiral in the sense that there is another, equally good bijection $\rho\beta\rho \neq \beta$, where ρ is left–right reflection (which acts by reversing the order of the children of each vertex); but it suffices to employ just one of these bijections.

2.2. **Avoidance.** The more formal way to think of an n-vertex tree is as a particular arrangement of n pairs of parentheses, where each vertex is represented by the pair of parentheses containing its children. For example, the tree

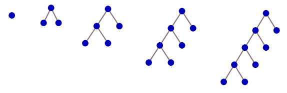
$T =$ 

is represented by `(()(()()))`. This is the word representation of this tree in the alphabet $\{$`(`,`)`$\}$. We do not formally distinguish between the graphical representation of a tree and the word representation, and it is the latter that is useful in manipulating trees algorithmically. (*Mathematica*'s pattern matching capabilities provide a convenient tool for working with trees represented as nested lists, so this is the convention used by TreePatterns.)

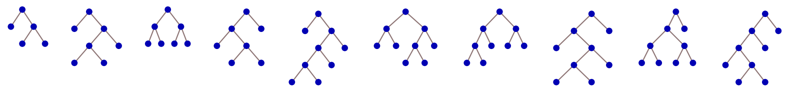
Informally, our concept of containment is as follows. A binary tree T *contains* t if there is a (contiguous, rooted, ordered) subtree of T that is a copy of t. For example, consider

$t =$ .

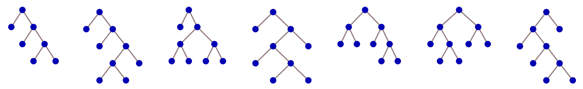
None of the trees



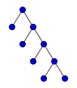
contains a copy of t, while each of the trees



contains precisely one copy of t, each of the trees



contains precisely two (possibly overlapping) copies of t, and the tree



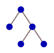
contains precisely three copies of t. This is a classification of binary trees with at most 5 leaves according to the number of copies of t.

We might formalize this concept with a graph theoretic definition as follows. Let t be a binary tree. A *copy* of t in T is a subgraph of T (obtained by removing vertices) that is isomorphic to t (preserving edge directions and the order of children). Naturally, T *avoids* t if the number of copies of t in T is 0.

An equivalent but much more useful definition is a language theoretic one, and to provide this we first distinguish a *tree pattern* from a tree.

By ‘tree pattern’, informally we mean a tree whose leaves are “blanks” that can be filled (matched) by any tree, not just a single vertex. More precisely, let $\Sigma = \{\texttt{(}, \texttt{)}\}$, and let L be the language on Σ containing (the word representation of) every binary tree. Consider a binary tree τ, and let t be the word on the three symbols $\texttt{(}, \texttt{)}, L$ obtained by replacing each leaf $\texttt{()}$ in τ by L. We call t the *tree pattern* of τ. This tree pattern naturally generates a language L_t on Σ, which we obtain by interpreting the word t as a product of the three languages $\texttt{(} = \{\texttt{(}\}$, $\texttt{)} = \{\texttt{)}\}$, L. Informally, L_t is the set of words that match t. We think of t and L_t interchangeably. (Note that a tree is a tree pattern matched only by itself.)

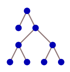
For example, let

$$\tau =$$  $$= \texttt{(()(()()))};$$

then the corresponding tree pattern is $t = \texttt{(}L\texttt{(}LL\texttt{))}$, and the language L_t consists of all trees of the form $\texttt{(}T\texttt{(}UV\texttt{))}$, where T, U, V are binary trees.

Let Σ^* denote the set of all finite words on Σ. The language $\Sigma^* L_t \Sigma^* \cap L$ is the set of all binary trees whose word has a subword in L_t. Therefore we say that a binary tree T *contains* the tree pattern t if T is in the language $\Sigma^* L_t \Sigma^* \cap L$. We can think of this language as a multiset, where a given tree T occurs with multiplicity equal to the number of ways that it matches $\Sigma^* L_t \Sigma^*$. Then the *number of copies* of t in T is the multiplicity of T in $\Sigma^* L_t \Sigma^* \cap L$.

Continuing the example from above, the tree

$$T =$$  $$= \texttt{(()((()())(()())))}$$

contains 2 copies of t since it matches $\Sigma^* L_t \Sigma^*$ in 2 ways: $\texttt{(}T\texttt{(}UV\texttt{))}$ with $T = \texttt{()}$ and $U = V = \texttt{(()())}$, and $\texttt{(()(}T\texttt{(}UV\texttt{)))}$ with $T = \texttt{(()())}$ and $U = V = \texttt{()}$.

Our notation distinguishes tree patterns from trees: Tree patterns are represented by lowercase variables, and trees are represented by uppercase variables. To be absolutely precise, we would graphically distinguish between terminal leaves $\texttt{()}$ of a tree and blank leaves L of a tree pattern, but this gets in the way of speaking about them as the same objects, which is rather convenient.

In Sections 4 and 5 we will be interested in taking the intersection $p \cap q$ of tree patterns p and q (by which we mean the intersection of the corresponding languages L_p and L_q). The intersection of two or more explicit tree patterns can be computed recursively: $p \cap L = p$, and $\texttt{(}p_l p_r\texttt{)} \cap \texttt{(}q_l q_r\texttt{)} = \texttt{(}(p_l \cap q_l)(p_r \cap q_r)\texttt{)}$.

2.3. **Generating functions.**

Our primary goal is to determine the number a_n of binary trees with n vertices that avoid a given binary tree pattern t, and more generally to determine the number $a_{n,k}$ of binary trees with n vertices and precisely k copies of t. Thus we consider two generating functions associated with t: the *avoiding generating function*

$$\mathrm{Av}_t(x) = \sum_{T \text{ avoids } t} x^{\text{number of vertices in } T} = \sum_{n=0}^{\infty} a_n x^n$$

and the *enumerating generating function*

$$\begin{aligned}\mathrm{En}_{L,t}(x,y) &= \sum_{T\in L} x^{\text{number of vertices in } T} y^{\text{number of copies of } t \text{ in } T}\\ &= \sum_{n=0}^{\infty}\sum_{k=0}^{\infty} a_{n,k}x^n y^k.\end{aligned}$$

The avoiding generating function is the special case $\mathrm{Av}_t(x) = \mathrm{En}_{L,t}(x,0)$.

Theorem 1. *$\mathrm{En}_{L,t}(x,y)$ is algebraic.*

The proof is constructive, so it enables us to compute $\mathrm{En}_{L,t}(x)$, and in particular $\mathrm{Av}_t(x)$, for explicit tree patterns. We postpone the proof until Section 4.2 to address a natural question that arises: Which trees have the same generating function? That is, for which pairs of binary tree patterns s and t are the n-leaf trees avoiding (or containing k copies of) these patterns equinumerous?

We say that s and t are *avoiding-equivalent* if $\mathrm{Av}_s(x) = \mathrm{Av}_t(x)$. We say they are *enumerating-equivalent* if the seemingly stronger condition $\mathrm{En}_{L,s}(x,y) = \mathrm{En}_{L,t}(x,y)$ holds. We can compute these equivalence classes explicitly by computing $\mathrm{Av}_t(x)$ and $\mathrm{En}_{L,t}(x,y)$ for, say, all m-leaf binary tree patterns t. In doing this for binary trees with up to 7 leaves, one comes to suspect that these conditions are in fact the same.

Conjecture 2. *If s and t are avoiding-equivalent, then they are also enumerating-equivalent.*

In light of this experimental result, we focus attention in the remainder of the paper on classes of avoiding-equivalence, since conjecturally they are the same as classes of enumerating-equivalence.

3. Initial inventory and some special bijections

In this section we undertake an analysis of small patterns. We determine $\mathrm{Av}_t(x)$ for binary tree patterns with at most 4 leaves using methods specific to each. This allows us to establish the equivalence classes in this range.

3.1. 1-leaf trees.

There is only one binary tree pattern with a single leaf, namely

$$t = \bullet = L.$$

Every binary tree contains at least one vertex, so $\mathrm{Av}_t(x) = 0$. The number of binary trees with $2n-1$ vertices is C_{n-1}, so

$$\mathrm{En}_L(x) = x + x^3 + 2x^5 + 5x^7 + 14x^9 + 42x^{11} + \cdots = \sum_{n=1}^{\infty} C_{n-1}x^{2n-1}.$$

3.2. 2-leaf trees.

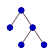
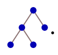
There is also only one binary tree pattern with precisely 2 leaves:

$t =$ $=$ `(LL)`.

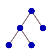
However, t is a fairly fundamental structure in binary trees; the only tree avoiding it is the 1-vertex tree `()`. Thus $\mathrm{Av}_t(x) = x$, and

$$\mathrm{En}_{L,t}(x,y) = \sum_{n=1}^{\infty} C_{n-1}x^{2n-1}y^{n-1} = \frac{1-\sqrt{1-4x^2y}}{2xy}.$$

3.3. **3-leaf trees.** There are $C_2 = 2$ binary trees with 3 leaves, and they are equivalent by left–right reflection:

 and .

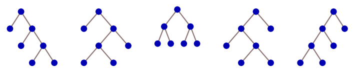
There is only one binary tree with n leaves avoiding

 $= \texttt{((}LL\texttt{)}L\texttt{)}$,

namely the "right comb" `(()(()(()(()···))))`. Therefore for these trees

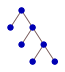
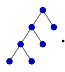
$$\mathrm{Av}_t(x) = x + x^3 + x^5 + x^7 + x^9 + x^{11} + \cdots = \frac{x}{1-x^2}.$$

3.4. **4-leaf trees.** Among 4-leaf binary trees we find more interesting behavior. There are $C_3 = 5$ such trees, pictured as follows.



They comprise 2 equivalence classes.

Class 4.1. The first equivalence class consists of the trees

$t_1 =$  and $t_5 =$ .

The avoiding generating function $\mathrm{Av}_t(x)$ for each of these trees satisfies

$$x^3 f^2 + (x^2 - 1)f + x = 0$$

because the number of n-leaf binary trees avoiding t_1 is the Motzkin number M_{n-1}:

$$\mathrm{Av}_t(x) = x + x^3 + 2x^5 + 4x^7 + 9x^9 + 21x^{11} + \cdots = \sum_{n=1}^{\infty} M_{n-1} x^{2n-1}.$$

This fact is presented by Donaghey and Shapiro [3] as their final example of objects counted by the Motzkin numbers. They provide a bijective proof which we reformulate here. Specifically, there is a natural bijection between the set of n-leaf binary trees avoiding t_1 and the set of Motzkin paths of length $n-1$ — paths from $(0,0)$ to $(n-1,0)$ composed of steps $\langle 1,-1\rangle$, $\langle 1,0\rangle$, $\langle 1,1\rangle$ that do not go below the x-axis. We represent a Motzkin path as a word on $\{-1,0,1\}$ encoding the sequence of steps under $\langle 1,\Delta y\rangle \mapsto \Delta y$. The bijection is as follows.

Recall the map β from Section 2.1. To obtain the Motzkin path associated with a binary tree T avoiding t_1:

(1) Let $T' = \beta(T)$. No vertex in T' has more than 2 children, since

$\beta(t_1) =$

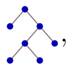
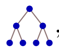
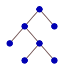
and T avoids t_1.

(2) Create a word w on $\{-1,0,1\}$ by traversing T' in depth-first order (i.e., for each subtree visit first the root vertex and then its children trees in order); for each vertex, record 1 less than the number of children of that vertex.

(3) Delete the last letter of w (which is -1).

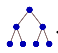
The resulting word contains the same number of -1s and 1s, and every prefix contains at least as many 1s as -1s, so it is a Motzkin path. The steps are easily reversed to provide the inverse map from Motzkin paths to binary trees avoiding t_1. (For the larger context of this bijection, see Stanley's presentation leading up to Theorem 5.3.10 [14].)

Class 4.2. The second equivalence class consists of the three trees

$t_2 =$ , $t_3 =$ , and $t_4 =$ 

and provides the smallest example of nontrivial equivalence. Symmetry gives $\mathrm{Av}_{t_2}(x) = \mathrm{Av}_{t_4}(x)$. To establish $\mathrm{Av}_{t_2}(x) = \mathrm{Av}_{t_3}(x)$, for each of these trees t we give a bijection between n-leaf binary trees avoiding t and binary words of length $n-2$. By composing these two maps we obtain a bijection between trees avoiding t_2 and trees avoiding t_3.

First consider

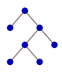
$t_3 =$ .

If T avoids t_3, then no vertex of T has four grandchildren; that is, at most one of a vertex's children has children of its own. This implies that at each generation at most one vertex has children. Since there are two vertices at each generation after the first, the number of such n-leaf trees is 2^{n-2} for $n \geq 2$:

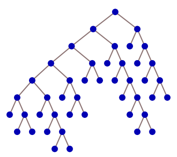
$$\mathrm{Av}_{t_3}(x) = x + x^3 + 2x^5 + 4x^7 + 8x^9 + 16x^{11} + \cdots = x + \sum_{n=2}^{\infty} 2^{n-2} x^{2n-1} = \frac{x(1-x^2)}{1-2x^2}.$$

Form a word $w \in \{0,1\}^{n-2}$ corresponding to T by letting the ith letter be 0 or 1 depending on which vertex (left or right) on level $i+1$ has children.

Now consider

$t_2 =$ .

A "typical" binary tree avoiding t_2 looks like



and is determined by the length of its spine and the length of each arm. Starting from the root, traverse the internal vertices of a tree T avoiding t_2 according to the following rule. Always move to the right child of a vertex when the right child is an internal vertex, and if the right child is a leaf then move to the highest unvisited internal spine vertex. By recording 0 and 1 for left and right movements in this traversal, a word w on $\{0,1\}$ is produced that encodes T uniquely. We have $|w| = n-2$ since we obtain one symbol from each internal vertex except the root. Since every word w corresponds to an n-leaf binary tree avoiding t_2, there are 2^{n-2} such trees.

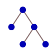
More formally, let ω be a map from binary trees to binary words defined by $\omega(\texttt{(}T_l T_r\texttt{)}) = \kappa_1(T_r)\kappa_0(T_l)$, where

$$\kappa_i(T) = \begin{cases} \epsilon & \text{if } T = \texttt{()} \\ i\,\omega(T) & \text{otherwise.} \end{cases}$$

Then the word corresponding to T is $w = \omega(T)$.

For the inverse map ω^{-1}, begin with the word $\texttt{(}lr\texttt{)}$. Then read w left to right. When the symbol 1 is read, replace the existing r by $\texttt{(()}r\texttt{)}$; when 0 is read, replace the existing r by $\texttt{()}$ and the existing l by $\texttt{(}lr\texttt{)}$. After the entire word is read, replace the remaining l and r with $\texttt{()}$. One verifies that T has n leaves. The tree T avoids t_2 because the left child of an r vertex never has children of its own.

3.5. **Bijections to Dyck words.** In Section 2.2 we assigned a word on the alphabet $\{\texttt{(}, \texttt{)}\}$ to each tree. In this section we use a slight variant of this word that is more widely used in the literature. This is the *Dyck word* on the alphabet $\{0, 1\}$, which differs from the aforementioned word on $\{\texttt{(}, \texttt{)}\}$ in that the root vertex is omitted. For example, the Dyck word of



is 01001011. Omitting the root allows consistency with the definition of a Dyck word as a word consisting of n 0s and n 1s such that no prefix contains more 1s than 0s. It is mnemonically useful to think of the letters in the Dyck word as the directions (down or up) taken along the edges in the depth-first traversal of the tree.

Because trees and Dyck words are essentially the same objects, one expects questions about pattern avoidance in trees to have an interpretation as questions about pattern avoidance in Dyck words. Specifically, the set of trees avoiding a certain tree pattern corresponds to the set of Dyck words avoiding a (not necessarily contiguous) "word pattern". This is simply a consequence of the bijection between trees and Dyck words.

However, in some cases there is a stronger relationship: The set of trees avoiding a certain pattern is in natural bijection to the set of Dyck words avoiding a certain contiguous subword. This relationship is the subject of the current section, in which we give several such bijections. For each equivalence class of trees we will be content with one bijection to Dyck words, although in many cases there are several.

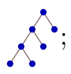
Notes on sequences counting Dyck words avoiding a subword have been contributed by David Callan and Emeric Deutsch to the Encyclopedia of Integer Sequences [13]. The subject appears to have begun with Deutsch [2, Section 6.17], who enumerated Dyck words according to the number of occurrences of the subword 100. Sapounakis, Tasoulas, and Tsikouras [12] have considered additional subwords. Via the bijections described below, their results provide additional derivations of the generating functions $\text{Av}_t(x)$.

In Section 3.4 we observed that trees avoiding a tree pattern in class 4.1 are in bijection to Motzkin paths. From here it is easy to establish a bijection to Dyck words avoiding the subword 000. Simply apply the morphism

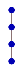
$$1 \to 001,\ 0 \to 01,\ -1 \to 1$$

to the Motzkin path. Clearly the resulting word has no instance of 000, and it is a Dyck word because 1 and -1 occur in pairs in a Motzkin path, with every prefix containing at least as many 1s as -1s.

A different and more direct bijection to Dyck words avoiding 000 (and the one that we will generalize to other tree patterns) can be obtained as follows. Consider the 4-leaf binary tree

$t_5 =$  ;

then

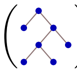
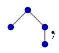
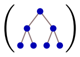
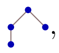
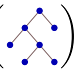
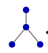
$\beta(t_5) =$  ,

whose Dyck word is 000111. The map β of Section 2.1 preserves a certain feature of any binary tree T containing t_5: $\beta(T)$ contains a sequence of four vertices in which each of the lower three is the first child of the previous. In other words, let T be a tree obtained by replacing the leaves of t_5 with any binary trees. Then $\beta(T)$ is obtained from $\beta(t_5)$ by adding vertices as either children or right siblings — never as left siblings — to those in $\beta(t_5)$. Therefore 000 is characteristic of t_5 in the sense that T contains t_5 if and only if the Dyck word of $\beta(T)$ contains 000.

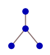
And of course this bijection works for any left comb (i.e., class $m.1$ for all m): The binary trees avoiding the m-leaf left comb are in bijection to Dyck words avoiding 0^{m-1}.

In general there is a natural bijection between n-leaf binary trees avoiding t and $(n-1)$-Dyck words avoiding w whenever w is a characteristic feature of $\beta(t)$ (that is, some feature of the tree that is preserved locally by β).

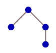
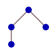
For example, for the binary trees in class 4.2 we have

$\beta\Big($  $\Big) =$  , $\beta\Big($  $\Big) =$  , $\beta\Big($  $\Big) =$  .

Which of these patterns have a bijection to Dyck words? Consider the third tree,

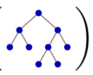
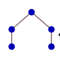
 $= 001011.$

While it is true that any tree containing this tree must contain 001, the converse is not true, so this tree does not admit a bijection to Dyck words. However, the two trees

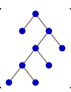
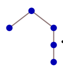
 $= 010011$ and  $= 001101$

contain the word 100 and its reverse complement 110 respectively, and containing one of these subwords is a necessary and sufficient condition for the corresponding tree to contain the respective tree pattern. Thus binary trees avoiding a tree pattern in class 4.2 are in bijection to Dyck words avoiding 100.

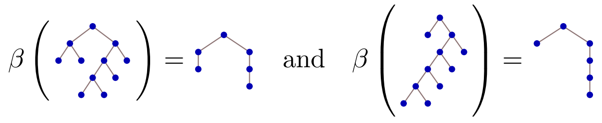
Bijections for other patterns can be found similarly. Binary trees avoiding a tree pattern in class 5.2 are in bijection to Dyck words avoiding 1100, via

$\beta\Big($  $\Big) =$  .

Class 5.3 corresponds to 1000 via

$\beta\Big($  $\Big) =$  .

Class 6.3 corresponds to 11000 and class 6.6 to 10000 via



respectively.

It is apparent that results of this kind involve "two-pronged" trees because avoidance for these trees corresponds to a local condition on Dyck words. It should not be surprising then that not all equivalence classes of binary trees have a corresponding Dyck word class. For example, classes 6.2, 6.4, 6.5, and 6.7 do not. The lack of a Dyck word class can be proven in each case by exhibiting an n such that the number of n-leaf binary trees avoiding t is not equal to the number of $(n-1)$-Dyck words avoiding w for all w; only a finite amount of computation is required because all C_{n-1} $(n-1)$-Dyck words avoid w for $|w| > 2(n-1)$. For example, $n = 8$ suffices for classes 6.2 and 6.5.

4. Algorithms

In this section we provide algorithms for computing algebraic equations satisfied by $\mathrm{Av}_t(x)$, $\mathrm{En}_{L,t}(x, y)$, and the more general $\mathrm{En}_{L,p_1,\ldots,p_k}(x_L, x_{p_1}, \ldots, x_{p_k})$ defined in Section 4.3. Computing $\mathrm{Av}_t(x)$ or $\mathrm{En}_{L,t}(x, y)$ for all m-leaf binary tree patters t allows one to automatically determine the equivalence classes given in the appendix.

We draw upon the notation introduced in Section 2.2. In particular, the intersection $p \cap p'$ of two tree patterns plays a central role. Recall that L_p is the set of trees matching p at the top level.

4.1. **Avoiding a single tree.**

Fix a binary tree pattern t we wish to avoid. For a given tree pattern p, we will make use of the generating function

$$\mathrm{weight}(p) = \mathrm{weight}(L_p) := \sum_{T \in L_p} \mathrm{weight}(T),$$

where

$$\mathrm{weight}(T) = \begin{cases} x^{\text{number of vertices in } T} & \text{if } T \text{ avoids } t \\ 0 & \text{if } T \text{ contains } t. \end{cases}$$

The case $t = L$ was covered in Section 3.1, so we assume $t \neq L$. Then $t = (t_l t_r)$ for some tree patterns t_l and t_r. Since $(T_l T_r)$ matches t precisely when T_l matches t_l and T_r matches t_r, we have

$$(1) \quad \mathrm{weight}((p_l p_r)) = \\ x \cdot \big(\mathrm{weight}(p_l) \cdot \mathrm{weight}(p_r) - \mathrm{weight}(p_l \cap t_l) \cdot \mathrm{weight}(p_r \cap t_r)\big).$$

The coefficient x is the weight of the root vertex of $(p_l p_r)$ that we destroy in separating this pattern into its two subpatterns.

We now construct a polynomial (with coefficients that are polynomials in x) that is satisfied by $\mathrm{Av}_t(x) = \mathrm{weight}(L)$, the weight of the language of binary trees. The algorithm is as follows.

Begin with the equation

$$\mathrm{weight}(L) = \mathrm{weight}(()) + \mathrm{weight}((LL)).$$

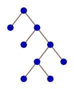
The variable weight($\texttt{(}LL\texttt{)}$) is "new"; we haven't yet written it in terms of other variables. So use Equation (1) to rewrite weight($\texttt{(}LL\texttt{)}$). For each expression weight($p \cap p'$) that is introduced, we compute the intersection $p \cap p'$. This allows us to write weight($p \cap p'$) as weight(q) for some pattern q that is simply a word on $\{\texttt{(}, \texttt{)}, L\}$ (i.e., does not contain the $\cap$ operator).

For each new variable weight(q), we obtain a new equation by making it the left side of Equation (1), and then as before we eliminate $\cap$ by explicitly computing intersections.

We continue in this manner until there are no new variables produced. This must happen because $\text{depth}(p \cap p') = \max(\text{depth}(p), \text{depth}(p'))$, so since there are only finitely many trees that are shallower than t, there are only finitely many variables in this system of polynomial equations.

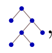
Finally, we compute a Gröbner basis for the system in which all variables except weight($\texttt{()}$) $= x$ and weight(L) $= \text{Av}_t(x)$ are eliminated. This gives a single polynomial equation in these variables, establishing that $\text{Av}_t(x)$ is algebraic.

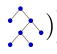
Let us work out an example. We use the graphical representation of tree patterns with the understanding that the leaves are blanks. Consider the tree pattern

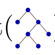
$t =$  $= \texttt{(}L\texttt{(}L\texttt{((}LL\texttt{)}L\texttt{)))}$

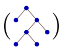
from class 5.2. The first equation is

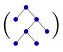
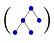
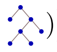
weight($\bullet$) $= x +$ weight( ).

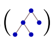
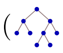
We have $t_l = \bullet$ and $t_r =$ , so Equation (1) gives

weight( ) $= x \cdot \big($weight($\bullet$) $\cdot$ weight($\bullet$) $-$ weight($\bullet \cap \bullet$) $\cdot$ weight($\bullet\, \cap$ )$\big)$

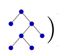
$= x \cdot \big($weight($\bullet$)^2 $-$ weight($\bullet$) $\cdot$ weight()$\big)$

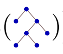
since $L \cap p = p$ for any tree pattern p. The variable weight() $=$ weight(t_r) is new, so we put it into Equation (1):

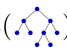
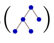
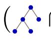
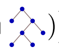
weight() $= x \cdot \big($weight($\bullet$) $\cdot$ weight() $-$ weight($\bullet \cap \bullet$) $\cdot$ weight($\cap$ )$\big)$

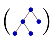
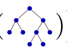
$= x \cdot \big($weight($\bullet$) $\cdot$ weight() $-$ weight($\bullet$) $\cdot$ weight()$\big)$.

There are two new variables:

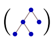
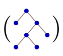
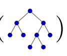
weight( ) $= x \cdot \big($weight( ) $\cdot$ weight($\bullet$) $-$ weight($\cap\, \bullet$) $\cdot$ weight($\bullet\, \cap$ )$\big)$

$= x \cdot \big($weight( ) $\cdot$ weight($\bullet$) $-$ weight( ) $\cdot$ weight()$\big)$;

weight() $= x \cdot \big($weight( ) $\cdot$ weight() $-$ weight($\cap\, \bullet$) $\cdot$ weight( $\cap$ )$\big)$

$= x \cdot \big($weight( ) $\cdot$ weight() $-$ weight( ) $\cdot$ weight()$\big)$.

We have no new variables, so we eliminate the four auxiliary variables

weight( ), weight(), weight(), weight()

from this system of five equations to obtain

x^3 weight($\bullet$)$^2 - (x^2-1)^2$ weight($\bullet$) $- x\,(x^2-1) = 0.$

4.2. **Enumerating with respect to a single tree.** To prove Theorem 1, we make a few modifications in order to compute $\mathrm{En}_{L,t}(x, y)$ instead of $\mathrm{Av}_t(x)$. Again

$$\mathrm{weight}(p) := \sum_{T \in L_p} \mathrm{weight}(T),$$

but now $\mathrm{weight}(T) = x^{\text{number of vertices in } T} y^{\text{number of copies of } t \text{ in } T}$ for all T. We modify Equation (1) to become

$$\text{(2)} \quad \mathrm{weight}(\texttt{(}p_l p_r\texttt{)}) = \\ x \cdot \big(\mathrm{weight}(p_l) \cdot \mathrm{weight}(p_r) + (y-1) \cdot \mathrm{weight}(p_l \cap t_l) \cdot \mathrm{weight}(p_r \cap t_r)\big)$$

since in addition to accounting for the trees that avoid t we also account for those that match t, in which case y is contributed.

The rest of the algorithm carries over unchanged, and we obtain a polynomial equation in x, y, and $\mathrm{En}_{L,t}(x, y) = \mathrm{weight}(L)$.

4.3. **Enumerating with respect to multiple trees.** A more general question is the following. Given several binary tree patterns $p_1, \dots, p_k$, what is the number $a_{n_0,n_1,\dots,n_k}$ of binary trees containing precisely n_0 vertices, n_1 copies of p_1, …, n_k copies of p_k? We consider the enumerating generating function

$$\begin{aligned}\mathrm{En}_{L,p_1,\dots,p_k}(x_L, x_{p_1}, \dots, x_{p_k}) &= \sum_{T \in L} x_L^{\alpha_0} x_{p_1}^{\alpha_1} \cdots x_{p_k}^{\alpha_k} \\ &= \sum_{n_0=0}^{\infty} \sum_{n_1=0}^{\infty} \cdots \sum_{n_k=0}^{\infty} a_{n_0,n_1,\dots,n_k} x_L^{n_0} x_{p_1}^{n_1} \cdots x_{p_k}^{n_k},\end{aligned}$$

where $p_0 = L$ and α_i is the number of copies of p_i in T. (We need not assume that the p_i are distinct.) This generating function can be used to obtain information about how correlated a family of tree patterns is. We have the following generalization of Theorem 1.

Theorem 3. *$\mathrm{En}_{L,p_1,\dots,p_k}(x_L, x_{p_1}, \dots, x_{p_k})$ is algebraic.*

Keeping track of multiple tree patterns $p_1, \dots, p_k$ is not much more complicated than handling a single pattern, and the algorithm for doing so has the same outline. Let

$$\mathrm{weight}(p) := \sum_{T \in L_p} \mathrm{weight}(T)$$

with

$$\mathrm{weight}(T) = x_L^{\alpha_0} x_{p_1}^{\alpha_1} \cdots x_{p_k}^{\alpha_k},$$

where again α_i is the number of copies of p_i in T. Let $d = \max_{1 \le i \le k} \mathrm{depth}(p_i)$. First we describe what to do with each new variable $\mathrm{weight}(q)$ that arises. The approach used is different than that for one tree pattern; in particular, we do not make use of intersections. Consequently, it is less efficient.

Let l be the number of leaves in q. If T is a tree matching q, then for each leaf L of q there are two possibilities: Either L is matched by a terminal vertex `()` in T, or L is matched by a tree matching `(`LL`)`. For each leaf we make this choice independently, thus partitioning the language L_q into 2^l disjoint sets represented by 2^l tree patterns that are disjoint in the sense that each tree matching q matches precisely one of these patterns. For example, partitioning the pattern (*LL*) into 2^2 patterns gives

$$\begin{aligned}\text{weight}((LL)) = \text{weight}((()())) + \\ \text{weight}((()(LL))) + \text{weight}(((LL)())) + \text{weight}(((LL)(LL))).\end{aligned}$$

We need an analogue of Equation (2) for splitting a pattern $(p_l p_r)$ into the two subpatterns p_l and p_r. For this, examine each of the 2^l patterns that arose in partitioning q. For each pattern $p = (p_l p_r)$ whose language is infinite (that is, the word p contains the symbol L) and has depth$(p) \geq d$, rewrite

$$\text{weight}(p) = \text{weight}(p_l) \cdot \text{weight}(p_r) \cdot \prod_{\substack{0 \leq i \leq k \\ p \text{ matches } p_i}} x_{p_i},$$

where 'p matches p_i' means that every tree in L_p matches p_i (so $L_p \subset L_{p_i}$). If L_p is infinite but depth$(p) < d$, keep weight(p) intact as a variable.

Finally, for all tree patterns p whose language is finite (i.e., p is a tree), rewrite

$$\text{weight}(p) = \prod_{0 \leq i \leq k} x_{p_i}^{\text{number of copies of } p_i \text{ in } p}.$$

The algorithm is as follows. As before, begin with the equation

$$\text{weight}(L) = \text{weight}(()) + \text{weight}((LL)).$$

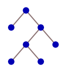
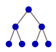
At each step, take each new variable weight(q) and obtain another equation by performing the procedure described: Write it as the sum of 2^l other variables, split the designated patterns into subpatterns, and explicitly compute the weights of any trees appearing. Continue in this manner until there are no new variables produced; this must happen because we break up weight(p) whenever depth$(p) \geq d$, so there are only finitely many possible variables. Eliminate from this system of polynomial equations all but the $k+2$ variables weight$(()) = x_L$, $x_{p_1}, \ldots, x_{p_k}$, and weight$(L) = \text{En}_{L,p_1,\ldots,p_k}(x_L, x_{p_1}, \ldots, x_{p_k})$ to obtain a polynomial equation satisfied by $\text{En}_{L,p_1,\ldots,p_k}(x_L, x_{p_1}, \ldots, x_{p_k})$.

5. Replacement bijections

In this section we address the question of providing systematic bijective proofs of avoiding-equivalence. Given two equivalent binary tree patterns s and t, we would like to produce an explicit bijection between binary trees avoiding s and binary trees avoiding t. It turns out that this can often be achieved by structural replacements on trees. We start by describing an example in full, and later generalize.

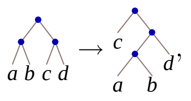
5.1. An example replacement bijection.

Consider the trees

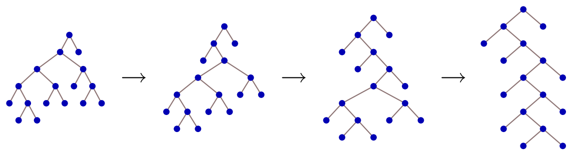
$t_2 =$  and $t_3 =$ 

in class 4.2. The idea is that since n-leaf trees avoiding t_2 are in bijection to n-leaf trees avoiding t_3, then somehow swapping all occurrences of these two tree patterns should produce a bijection. However, since the patterns may overlap, it is necessary to specify an order in which to perform the replacements. A natural order is to start with the root and work down the tree. More precisely, a *top-down replacement* is a restructuring of a tree T in which we iteratively apply a set of transformation rules to subtrees of T, working downward from the root.

Take the replacement rule to be

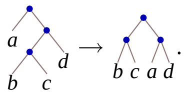


where the variables represent trees attached at the leaves, rearranged according to the permutation 3124. Begin at the root: If T itself matches the left side of the rule, then we restructure T according to the rule; if not, we leave T unchanged. Then we repeat the procedure on the root's (new) children, then on their children, and so on, so that each vertex in the tree is taken to be the root of a subtree which is possibly transformed by the rule. For example,



shows the three replacements required to compute the image (on the right) of a tree avoiding t_2. The resulting tree avoids t_3.

This top-down replacement is invertible. The inverse map is a *bottom-up replacement* with the inverse replacement rule,

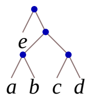
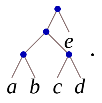


Rather than starting at the root and working down the tree, we apply this map by starting at the leaves and working up the tree.

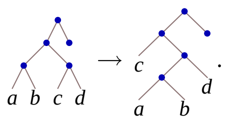
We now show that the top-down replacement is in fact a bijection from trees avoiding t_2 to trees avoiding t_3. It turns out to be the same bijection given in Section 3.4 via words in $\{0, 1\}^{n-2}$.

Assume T avoids t_2; we show that the image of T under the top-down replacement avoids t_3. It is helpful to think of T as broken up into (possibly overlapping) "spheres of influence" — subtrees which are maximal with respect to the replacement rule in the sense that performing the top-down replacement on the subtree does not introduce instances of the relevant tree patterns containing vertices outside of the subtree. It suffices to consider each sphere of influence separately. A natural focal point for each sphere of influence is the highest occurrence of t_3. We verify that restructuring this t_3 to t_2 under the top-down replacement produces no t_3 above, at, or below the root of the new t_2 in the image of T.

above: Since t_3 has depth 2, t_3 can occur at most one level above the root of the new t_2 while overlapping it. Thus it suffices to consider all subtrees with t_3 occurring at level 1. There are two cases,

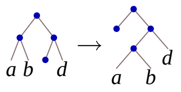
 and 

The first case does not avoid t_2, so it does not occur in T. The second case may occur in T. However, we do not want the subtree itself to match t_3 (because we assume that the t_3 at level 1 is the highest t_3 in this sphere of influence), so we must have $e =$ `()`. Thus this subtree is transformed by the top-down replacement as



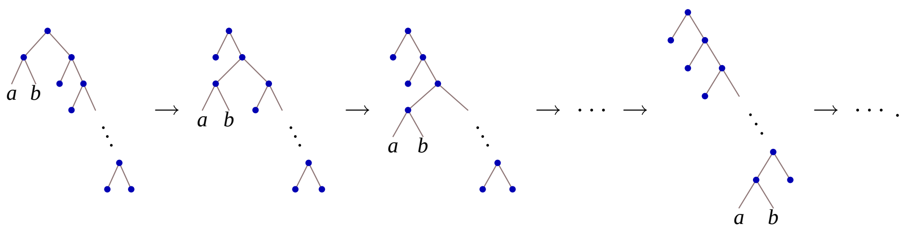
The image does not match t_3 at the root, so t_3 does not appear above the root of the new t_2.

at: Since T avoids t_2, every subtree in T matching t_3 in fact matches the pattern `((`LL`)(()`L`))`. Such a subtree is restructured as



under the replacement rule, and the image does not match t_3 (because $c =$ `()` is terminal). Therefore the new t_2 cannot itself match t_3.

below: A general subtree matching t_3 and avoiding t_2 is transformed as



Clearly t_3 can only occur in the image at or below the subtree `(`ab`)`. However, since `(`ab`)` is preserved by the replacement rule, any transformations on `(`ab`)` can be considered independently. That is, `(`ab`)` is the top of a different sphere of influence, so we need not consider it here. We conclude that t_3 does not occur below the root of the new t_2.

If we already knew that t_2 and t_3 are equivalent (for example, by having computed $\mathrm{Av}_t(x)$ as in Section 4.1), then we have now obtained a bijective proof of their equivalence. Otherwise, it remains to show that if T avoids t_3, then performing the bottom-up replacement produces a tree that avoids t_2; this can be accomplished similarly.

5.2. **General replacement bijections.** A natural question is whether for any two equivalent binary tree patterns s and t there exists a sequence of replacement bijections and left–right reflections that establishes their equivalence. For tree patterns of at most 7 leaves the answer is "Yes", which perhaps suggests that these maps suffice in general.

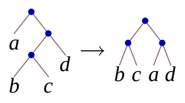
Conjecture 4. *If s and t are equivalent, then there is a sequence of top-down replacements, bottom-up replacements, and left–right reflections that produces a bijection from binary trees avoiding s to binary trees avoiding t.*

In this section we discuss qualitative results regarding this conjecture.

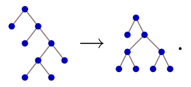
Given two m-leaf tree patterns s and t, one can ask which permutations of the leaves give rise to a top-down replacement that induces a bijection from trees avoiding s to trees avoiding t. Most permutations are not viable. Candidate permutations can be found experimentally by simply testing all $m!$ permutations of leaves on a set of randomly chosen binary trees avoiding s; one checks that the image avoids t and that composing the top-down replacement with the inverse bottom-up replacement produces the original tree. This approach is feasible for small m, but it is slow and does not provide any insight into why certain trees are equivalent. A question unresolved at present is to efficiently find all such bijections. The naive method was used to find the examples in this section.

Once a candidate bijection is found, it can be proved in a manner similar to Section 5.1. Although we do not attempt here to fully generalize that argument, the following two examples provide an indication of the issues encountered in the 'above' and 'below' cases of the general setting.

As an example of what can go wrong in the 'above' case, consider the rule



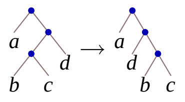
given by the permutation 2314 on 4-leaf binary trees $s = t_3$ and $t = t_2$. This rule does not induce a top-down replacement bijection from trees avoiding t_3 to trees avoiding t_2. One obstruction is the mapping



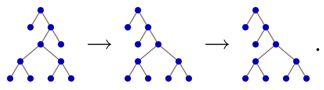
The initial tree avoids t_3. However, it contains t_2, and replacing this t_2 with t_3 completes another t_2. The final tree does not avoid t_2.

In general, for the 'above' case it suffices to check all trees avoiding s of a certain depth where the highest t begins at level depth$(t) - 1$. A bound on the depth is possible since for a given set of replacement rules there is a maximum depth at which the structure of a subtree can affect the top depth$(t) + 1$ levels of the image. If, after performing the top-down replacement on these trees, no t appears above level depth$(t) - 1$, then t can never appear above the root of the highest t in the subtree.

An example of what can go wrong in the 'below' case is provided by the rule



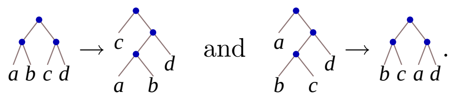
given by the permutation 1423 on 4-leaf binary trees $s = t_1$ and $t = t_2$. This rule does not induce a top-down replacement bijection from trees avoiding t_1 to trees avoiding t_2. (Indeed, these patterns are not equivalent.) One obstruction is the mapping



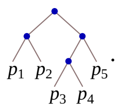
The initial tree avoids t_1. However, it matches t_2, and replacing this t_2 with t_1 produces a copy of the intersection $t_1 \cap t_2$ in the intermediate step. The t_2 is then replaced by t_1, but this does not change the tree because $t_1 \cap t_2$ is fixed by the rule. Therefore the final tree does not avoid t_2.

A general proof for the 'below' case must take into account all ways of producing, below the root of the highest t in a subtree, a copy t of that is not broken by further replacements.

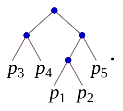
We now return briefly to the replacement rule of Section 5.1 to mention that a minor modification produces a bijection on the full set of binary trees. Namely, take the two replacement rules



Again we perform a top-down replacement, now using both rules together. That is, if a subtree matches the left side of either rule, we restructure it according to that rule. Of course, it can happen that a particular subtree matches both replacement rules, resulting in potential ambiguity; in this case which do we apply? Well, if both rules result in the same transformation, then it does not matter, and indeed with our present example this is the case. To show this, it suffices to take the intersection $t_2 \cap t_3$ of the two left sides and label the leaves to represent additional branches that may be present:



Now we check that applying each of the two replacement rules to this tree produces the same labeled tree, namely



Therefore we need not concern ourselves with which rule is applied to a given subtree that matches both. Since the replacement rules agree on their intersection, the top-down replacement is again invertible and is therefore a bijection from the set of binary trees to itself. By the examination of cases in Section 5.1, this bijection is an extension of the bijection between binary trees avoiding t_2 and binary trees avoiding t_3.

Thus we may choose from two types of bijection when searching for top-down replacement bijections that prove avoiding-equivalence. The first type is from binary trees avoiding s to binary trees avoiding t, using one rule for the top-down direction and the inverse for the bottom-up direction; these bijections in general do not extend to bijections on the full set of binary trees. The second type is a bijection on the full set of binary trees, using both rules in each direction, that induces a bijection from binary trees avoiding s to binary trees avoiding t.

Empirically, each two-rule bijection that proves avoiding-equivalence for s and t induces a one-rule bijection proving this equivalence, as in the previous example. However, not all two-rule bijections, when restricted to one rule, become bijections that prove avoiding-equivalence; the rules used to discuss the 'above' and 'below' cases in this section are two counterexamples.

One benefit to searching for two-rule bijections is that requiring the two replacement rules to agree on the leaf-labeled intersection $s \cap t$ quickly prunes the set of candidate permutations. There is a tradeoff, however, because verifying a two-rule bijection is more complicated than verifying a one-rule bijection.

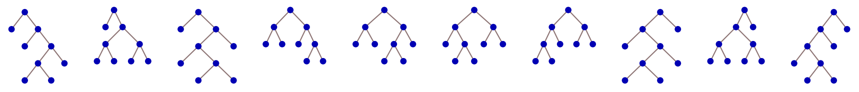
	t_2	t_3	t_4	t_6	t_7	t_8	t_9	t_{11}	t_{12}	t_{13}
t_2	—	14235		43125						
t_3		—	12534	31245					51234	
t_4		12453	—			41235				
t_6				—	12534	45123				
t_7				12453	—		45123			
t_8				34512		—	31245			
t_9					34512	23145	—			
t_{11}					13452			—	31245	
t_{12}		23451					12453	23145	—	
t_{13}							14532		13425	—

Table 1. Leaf permutations whose two-rule replacements prove avoiding-equivalence for pairs of trees in class 5.2.

Searching for two-rule bijections on 4-leaf binary tree patterns, one finds only the rules given by the permutation 3124 for $s = t_2$ and $t = t_3$, mentioned above, and their left–right reflections, given by 1342 for $s = t_4$ and $t = t_3$. (Note the asymmetry here: There is a top-down replacement bijection from trees avoiding t_2 to trees avoiding t_3 but not vice versa.)

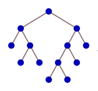
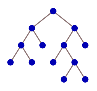
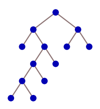
However, this does not account for all top-down replacement bijections for these patterns. The permutation 3142 for $s = t_2$ and $t = t_3$ and its left–right reflection 3142 for $s = t_4$ and $t = t_3$ provide one-rule bijections that do not extend to all binary trees.

Among equivalence classes of 5-leaf binary tree patterns, class 5.2 is the only class containing nontrivial equivalences. It consists of the ten trees t_2, t_3, t_4, t_6, t_7, t_8, t_9, t_{11}, t_{12}, and t_{13}, pictured as follows.



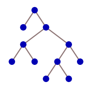
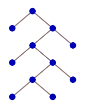
For each pair of trees in this class, Table 1 lists the leaf permutations that prove equivalence by a two-rule top-down replacement. It happens that there is at most one permutation for each pair in this class, although in general there may be more.

With this data, one might suspect that two-rule replacement bijections are sufficient to establish every equivalence class of binary tree patterns. In fact they are not. The smallest counterexample is class 7.15, which consists of the three trees

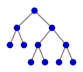
$t_{61} =$ , $t_{65} =$ , and $t_{81} =$ 

and their left–right reflections. Trees t_{81} and t_{61} are equivalent by the permutation 1456723, but no permutation of leaves produces a two-rule replacement bijection that establishes the equivalence of t_{65} to one of the others. However, the permutations 4561237 and 4571236 provide candidate one-rule bijections for t_{65} and t_{61}, and 2341675 provides a candidate bijection for t_{65} and t_{81}.

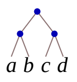
We conclude with a curious example in which two tree patterns can only be proven equivalent by a two-rule bijection that does not involve them directly. The trees

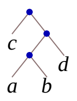
$t_7 =$  and $t_{11} =$ 

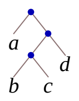
in class 6.5 are avoiding-equivalent by the permutation 126345, but neither

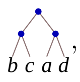
$t_{17} =$ 

nor its left–right reflection has an equivalence-proving permutation to t_7, t_{11}, or their left–right reflections. Thus, this equivalence cannot be established by a bijection that swaps 6-leaf tree patterns. However, it can be established by a bijection that swaps 4-leaf tree patterns: The previously mentioned bijection consisting of the two replacement rules



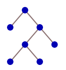
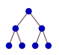
$\to$ 

and 

$\to$ 

,

induces a top-down replacement bijection from trees avoiding t_7 to trees avoiding t_{17}. The reason is that t_7 and t_{17} are formed by two overlapping copies of the class 4.2 trees

 and 

respectively, and that t_7 and t_{17} are mapped to each other under this bijection.

Acknowledgements

I thank Phillipe Flajolet for helping me understand some existing literature, and I thank Lou Shapiro for suggestions which clarified some points in the paper. Thanks to the referee for the reference to Stanley's book.

I am indebted to Elizabeth Kupin for much valuable feedback. Her comments greatly improved the exposition and readability of the paper. In addition, the idea of looking for one-rule bijections that do not extend to bijections on the full set of binary trees is hers, and this turned out to be an important generalization of the two-rule bijections I had been considering.

Appendix. Table of equivalence classes

This appendix lists equivalence classes of binary trees with at most 6 leaves. Left–right reflections are omitted for compactness. For each class we provide a polynomial equation satisfied by $f = \mathrm{En}_{L,t}(x, y)$; an equation satisfied by $\mathrm{Av}_t(x)$ is obtained in each case by letting $y = 0$.

The data was computed by the *Mathematica* package TreePatterns [11] using Singular via the interface package by Manuel Kauers and Viktor Levandovskyy [8]. Pre-computed data extended to 8-leaf binary trees is now also available in TreePatterns. The number of equivalence classes of m-leaf binary trees for $m = 1, 2, 3, \ldots$ is $1, 1, 1, 2, 3, 7, 15, 44, \ldots$.

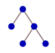
Class 1.1 (1 tree).

$$xyf^2 - f + xy = 0$$

Class 2.1 (1 tree).

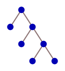
$$xyf^2 - f + x = 0$$

Class 3.1 (2 trees).



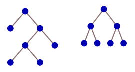
$$xyf^2 + \left(-x^2(y-1) - 1\right) f + x = 0$$

Class 4.1 (2 trees).



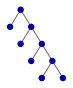
$$\left(xy - x^3(y-1)\right) f^2 + \left(-x^2(y-1) - 1\right) f + x = 0$$

Class 4.2 (3 trees).



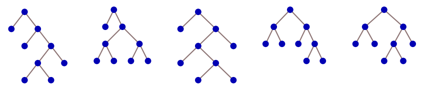
$$xyf^2 + \left(-2x^2(y-1) - 1\right) f + \left(x^3(y-1) + x\right) = 0$$

Class 5.1 (2 trees).



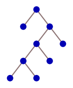
$$-x^4(y-1)f^3 + \left(xy - x^3(y-1)\right) f^2 + \left(-x^2(y-1) - 1\right) f + x = 0$$

Class 5.2 (10 trees).



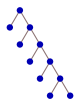
$$\left(xy - x^3(y-1)\right) f^2 + \left(x^2\left(x^2 - 2\right)(y-1) - 1\right) f + \left(x^3(y-1) + x\right) = 0$$

Class 5.3 (2 trees).



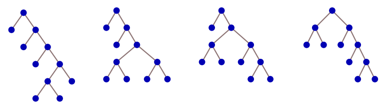
$$xyf^3 + \left(-3x^2(y-1) - 1\right) f^2 + \left(3x^3(y-1) + x\right) f - x^4(y-1) = 0$$

Class 6.1 (2 trees).



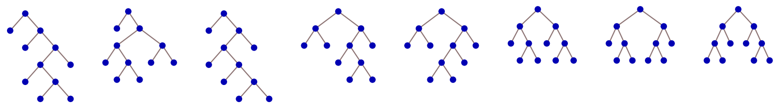
$$-x^5(y-1)f^4 - x^4(y-1)f^3 + \left(xy - x^3(y-1)\right) f^2 + \left(-x^2(y-1) - 1\right) f + x = 0$$

Class 6.2 (8 trees).



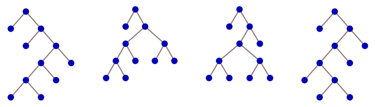
$$- x^4(y-1)f^3 + x\left(x^2\left(x^2 - 1\right)(y-1) + y\right) f^2 +$$
$$\left(x^2\left(x^2 - 2\right)(y-1) - 1\right) f + \left(x^3(y-1) + x\right) = 0$$

Class 6.3 (14 trees).



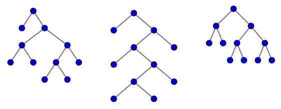
$$x\left(x^2\left(x^2-2\right)(y-1)+y\right)f^2+\left(2x^2\left(x^2-1\right)(y-1)-1\right)f+\left(x^3(y-1)+x\right)=0$$

Class 6.4 (8 trees).



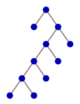
$$\left(xy-x^3(y-1)\right)f^3+\left(x^2\left(2x^2-3\right)(y-1)-1\right)f^2+$$
$$\left(-x^5(y-1)+3x^3(y-1)+x\right)f-x^4(y-1)=0$$

Class 6.5 (6 trees).



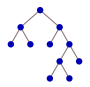
$$\left(xy-2x^3(y-1)\right)f^2+\left(x^2\left(3x^2-2\right)(y-1)-1\right)f+\left(-x^5(y-1)+x^3(y-1)+x\right)=0$$

Class 6.6 (2 trees).



$$-xyf^4+\left(4x^2(y-1)+1\right)f^3+\left(-6x^3(y-1)-x\right)f^2+4x^4(y-1)f-x^5(y-1)=0$$

Class 6.7 (2 trees).



$$x^4\left(x^2(y-1)-y\right)(y-1)f^3+\left(-2x^7(y-1)^2+x^5(y-1)(3y-2)-x^3(y-1)+xy\right)f^2+$$
$$\left(x^2\left(x^6(y-1)-3x^4(y-1)+x^2-2\right)(y-1)-1\right)f+\left(x^7(y-1)^2+x^3(y-1)+x\right)=0$$

References

[1] Nicolaas de Bruijn and B. J. M. Morselt, A note on plane trees, *Journal of Combinatorial Theory* **2** (1967) 27–34.

[2] Emeric Deutsch, Dyck path enumeration, *Discrete Mathematics* **204** (1999) 167–202.

[3] Robert Donaghey and Louis Shapiro, Motzkin numbers, *Journal of Combinatorial Theory, Series A* **23** (1977) 291–301.

[4] Philippe Flajolet and Robert Sedgewick, *Analytic Combinatorics*, Cambridge University Press, 2009.

[5] Philippe Flajolet, Paolo Sipala, and Jean-Marc Steyaert, Analytic variations on the common subexpression problem, *Lecture Notes in Computer Science: Automata, Languages, and Programming* **443** (1990) 220–234.

[6] Ian Goulden and David Jackson, An inversion theorem for cluster decompositions of sequences with distinguished subsequences, *Journal of the London Mathematical Society* (second series) **20** (1979) 567–576.

[7] Frank Harary, Geert Prins, and William Tutte, The number of plane trees, *Indagationes Mathematicae* **26** (1964) 319–329.

[8] Manuel Kauers and Viktor Levandovskyy, SINGULAR [a *Mathematica* package], available from `http://www.risc.uni-linz.ac.at/research/combinat/software/Singular/index.html`.

[9] Donald Knuth, *The Art of Computer Programming*, second edition, Volume 1: Fundamental Algorithms, Addison–Wesley, 1973.

[10] John Noonan and Doron Zeilberger, The Goulden–Jackson cluster method: extensions, applications, and implementations, *Journal of Difference Equations and Applications* **5** (1999) 355–377.

[11] Eric Rowland, TREEPATTERNS [a *Mathematica* package], available from the author's web site.

[12] Aris Sapounakis, Ioannis Tasoulas, and Panos Tsikouras, Counting strings in Dyck paths, *Discrete Mathematics* **307** (2007) 2909–2924.

[13] Neil Sloane, The Encyclopedia of Integer Sequences, `http://www.research.att.com/~njas/sequences`.

[14] Richard Stanley, *Enumerative Combinatorics* volume 2, Cambridge University Press, New York, 1999.

[15] Jean-Marc Steyaert and Philippe Flajolet, Patterns and pattern-matching in trees: an analysis, *Information and Control* **58** (1983) 19–58.

MATHEMATICS DEPARTMENT, TULANE UNIVERSITY, NEW ORLEANS, LA 70118, USA