Journal of Combinatorial Theory, Series A 117 (2010) 741–758

Contents lists available at ScienceDirect

Journal of Combinatorial Theory, Series A

www.elsevier.com/locate/jcta

Pattern avoidance in binary trees

Eric S. Rowland

Mathematics Department, Tulane University, New Orleans, LA 70118, USA

ARTICLE INFO

Article history:
Received 8 December 2008
Available online 6 March 2010

Keywords:
Pattern avoidance
Binary trees
Wilf equivalence
Algebraic generating functions
Dyck words

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. 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 [5,7], 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}

E-mail address: erowland@tulane.edu.

0097-3165/$ – see front matter
doi:10.1016/j.jcta.2010.03.004

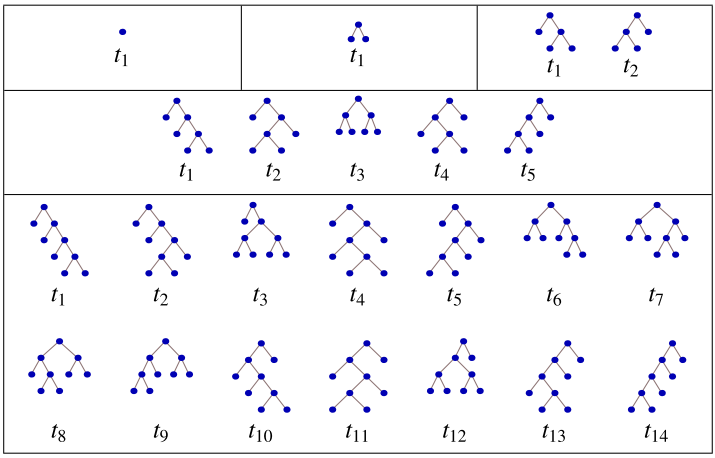


Fig. 1. The binary trees with at most 5 leaves.

In this paper we consider the analogous questions for plane trees. All trees in the paper are rooted and ordered.

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 Fig. 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 with each $(n-k)$-leaf binary tree, for all $1 \leqslant k \leqslant n-1$. 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 [12] 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 [4], 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 [3, 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 $i \geqslant 1$ 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 [8], 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 $n \geqslant 1$ 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. Appendix A 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 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. 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

$T =$ 

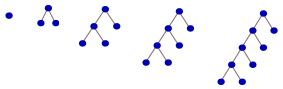
is represented by `(()(()()))`. This is the word representation of this tree in the alphabet $\{(,)\}$. We do not formally distinguish between the graphical representation of a tree and the word represen-

tation, 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.)

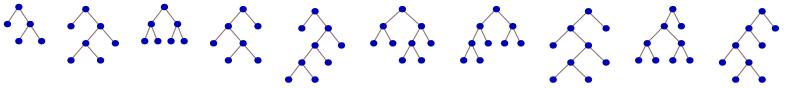
Informally, our concept of containment is as follows. A binary tree T *contains* t if there is a (contiguous, rooted, ordered) subtree of T that is a copy of t. For example, consider

$t =$  .

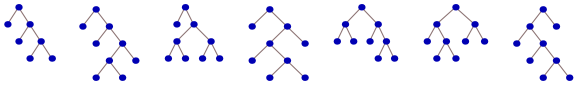
None of the trees



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 $\Sigma = \{(,)\}$, 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 $(,), L$ obtained by replacing each leaf $()$ 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 $(= \{(\},) = \{)\}$, 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

$\tau =$  $= (\,()\,(\,()\,()\,)\,)$;

then the corresponding tree pattern is $t = (\,L\,(\,LL\,)\,)$, and the language L_t consists of all trees of the form $(\,T\,(\,UV\,)\,)$, 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

$T =$  $= (\,()\,(\,(\,()\,)\,()\,)\,(\,()\,()\,)\,)$

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

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 $()$ 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 $(\,p_l p_r\,) \cap (\,q_l q_r\,) = ((p_l \cap q_l)(p_r \cap q_r))$.

2.2. 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

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

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

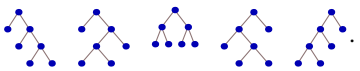
 $= ((LL)L)$,

namely the "right comb" $(()(()(()(()\cdots))))$. 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

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

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 [2] 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 $\langle 1, \Delta y\rangle \mapsto \Delta y$.

Let β be the usual bijection from n-leaf binary trees to n-vertex general trees that operates by contracting every rightward edge. 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 [11].)

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 $n \geqslant 2$:

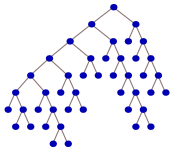
$$\mathrm{Av}_{t_3}(x) = x + x^3 + 2x^5 + 4x^7 + 8x^9 + 16x^{11} + \cdots = x + \sum_{n=2}^{\infty} 2^{n-2} x^{2n-1} = \frac{x(1-x^2)}{1-2x^2}.$$

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 $\omega((T_l T_r)) = \kappa_1(T_r)\kappa_0(T_l)$, where

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

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

For the inverse map ω^{-1}, begin with the word (lr). Then read w left to right. When the symbol 1 is read, replace the existing r by $(()r)$; when 0 is read, replace the existing r by $()$ and the existing l by (lr). After the entire word is read, replace the remaining l and r with $()$. 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

We mention that in some cases the set of trees avoiding a pattern is in bijection to the set of Dyck words avoiding a certain subword. For example, trees avoiding a pattern in class 4.1 are in bijection to Dyck words avoiding the subword 000. Recall β from Section 3.4. For t_5 we have

$\beta\left($  $\right) =$,

whose corresponding Dyck word is 000111, and β has the feature that T contains t_5 if and only if the Dyck word corresponding to $\beta(T)$ contains 000.

In general there is a 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 t_2 in class 4.2 we observe that

$\beta\left($  $\right) =$  $= 010011,$

and containing 100 as a subword is necessary and sufficient for the corresponding tree to contain t_2. Thus binary trees avoiding a pattern in class 4.2 are in bijection to Dyck words avoiding 100. Other bijections can be found similarly: Classes 5.2, 5.3, 6.3, and 6.6 correspond to the words 1100, 1000, 11000, and 10000.

Notes on sequences counting Dyck words avoiding a subword have been contributed by David Callan and Emeric Deutsch to the Encyclopedia of Integer Sequences [10]. The subject appears to have begun with Deutsch [1, Section 6.17], who enumerated Dyck words according to the number of occurrences of 100. Sapounakis, Tasoulas, and Tsikouras [9] have considered additional subwords. Via the bijections just described, their results provide additional derivations of the generating functions $\mathrm{Av}_t(x)$.

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 Appendix A.

We draw upon the notation introduced in Section 2.1. 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.

The *depth* of a vertex in a tree is the length of the minimal path to that vertex from the root, and $\mathrm{depth}(T)$ is the maximum vertex depth in the tree T.

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

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

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

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

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

For each new variable $\mathrm{weight}(q)$, we obtain a new equation by making it the left side of Eq. (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 $\mathrm{depth}(p \cap p') = \max(\mathrm{depth}(p), \mathrm{depth}(p'))$, 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 $\mathrm{weight}(())$ $= x$ and $\mathrm{weight}(L) = \mathrm{Av}_t(x)$ are eliminated. This gives a single polynomial equation in these variables, establishing that $\mathrm{Av}_t(x)$ 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

$t =$  $= (L\,(L\,(\,(LL)\,L)\,))$

from class 5.2. The first equation is

$$\text{weight}(\bullet) = x + \text{weight}(\text{  }).$$

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

$$\begin{aligned}\text{weight}(\text{  }) &= x\cdot\Big(\text{weight}(\bullet)\cdot\text{weight}(\bullet) - \text{weight}(\bullet\cap\bullet)\cdot\text{weight}\Big(\bullet\cap\text{  }\Big)\Big)\\ &= x\cdot\Big(\text{weight}(\bullet)^2 - \text{weight}(\bullet)\cdot\text{weight}\Big(\text{  }\Big)\Big)\end{aligned}$$

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

$$\begin{aligned}\text{weight}\Big(\text{  }\Big) &= x\cdot\Big(\text{weight}(\bullet)\cdot\text{weight}\Big(\text{  }\Big) - \text{weight}(\bullet\cap\bullet)\cdot\text{weight}\Big(\text{  }\Big)\Big)\\ &= x\cdot\Big(\text{weight}(\bullet)\cdot\text{weight}\Big(\text{  }\Big) - \text{weight}(\bullet)\cdot\text{weight}\Big(\text{  }\Big)\Big).\end{aligned}$$

There are two new variables:

$$\begin{aligned}\text{weight}\Big(\text{  }\Big) &= x\cdot\Big(\text{weight}(\text{  })\cdot\text{weight}(\bullet) - \text{weight}(\text{}\cap\bullet)\cdot\text{weight}\Big(\bullet\cap\text{  }\Big)\Big)\\ &= x\cdot\Big(\text{weight}(\text{  })\cdot\text{weight}(\bullet) - \text{weight}(\text{})\cdot\text{weight}\Big(\text{  }\Big)\Big);\\ \text{weight}\Big(\text{  }\Big) &= x\cdot\Big(\text{weight}(\text{  })\cdot\text{weight}\Big(\text{  }\Big) - \text{weight}(\text{}\cap\bullet)\cdot\text{weight}\Big(\text{  }\Big)\Big)\\ &= x\cdot\Big(\text{weight}(\text{  })\cdot\text{weight}\Big(\text{  }\Big) - \text{weight}(\text{})\cdot\text{weight}\Big(\text{  }\Big)\Big).\end{aligned}$$

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

$$\text{weight}(\text{  }),\ \text{weight}\Big(\text{  }\Big),\ \text{weight}\Big(\text{  }\Big),\ \text{weight}\Big(\text{}\Big)$$

from this system of five equations to obtain

$$x^3\text{weight}(\bullet)^2 - \big(x^2-1\big)^2\text{weight}(\bullet) - x\big(x^2-1\big) = 0.$$

4.2. Enumerating with respect to a single tree

To prove Theorem 1, we make a few modifications in order to compute $\text{En}_{L,t}(x,y)$ instead of $\text{Av}_t(x)$. Again

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

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

$$\begin{aligned}&\text{weight}\big(\,(\,p_l\,p_r\,)\,\big)\\ &\quad= x\cdot\big(\text{weight}(p_l)\cdot\text{weight}(p_r) + (y-1)\cdot\text{weight}(p_l\cap t_l)\cdot\text{weight}(p_r\cap t_r)\big)\end{aligned}\tag{2}$$

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 $p_1, \ldots, p_k$, what is the number $a_{n_0,n_1,\ldots,n_k}$ of binary trees containing precisely n_0 vertices, n_1 copies of $p_1, \ldots, n_k$ copies of p_k? We consider the enumerating generating function

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

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

Keeping track of multiple tree patterns $p_1, \ldots, p_k$ 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 $d = \max_{1 \leqslant i \leqslant k} \mathrm{depth}(p_i)$. 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

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

We need an analogue of Eq. (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 $\mathrm{depth}(p) \geqslant d$, rewrite

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

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 $\mathrm{depth}(p) < d$, keep $\mathrm{weight}(p)$ intact as a variable.

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

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

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

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

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) $\geqslant d$, so there are only finitely many possible variables. Eliminate from this system of polynomial equations all but the $k+2$ variables $\text{weight}(\,(\,)\,) = x_L$, $x_{p_1}, \ldots, x_{p_k}$, and $\text{weight}(L) = \text{En}_{L,p_1,\ldots,p_k}(x_L, x_{p_1}, \ldots, x_{p_k})$ 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.

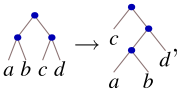
5.1. An example replacement bijection

Consider the trees

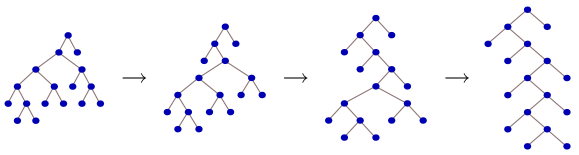
$t_2 =$  and $t_3 =$ 

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

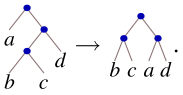


where the variables represent trees attached at the leaves, rearranged according to the permutation 3124. Begin at the root: If T itself matches the left side of the rule, then we restructure T according to the rule; if not, we leave T unchanged. Then we repeat the procedure on the root's (new) children, 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,

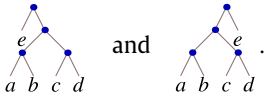


Rather than starting at the root and working down the tree, we apply this map by starting at the leaves and working up the tree.

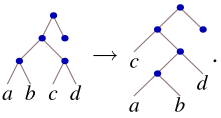
We now show that the top–down replacement is in fact a bijection from trees avoiding t_2 to trees avoiding t_3. It turns out to be the same bijection given in Section 3.4 via words in $\{0,1\}^{n-2}$.

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,

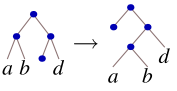


The first case does not avoid t_2, so it does not occur in T. The second case may occur in T. However, we do not want the subtree itself to match t_3 (because we assume that the t_3 at level 1 is the highest t_3 in this sphere of influence), so we must have $e = ()$. Thus this subtree is transformed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



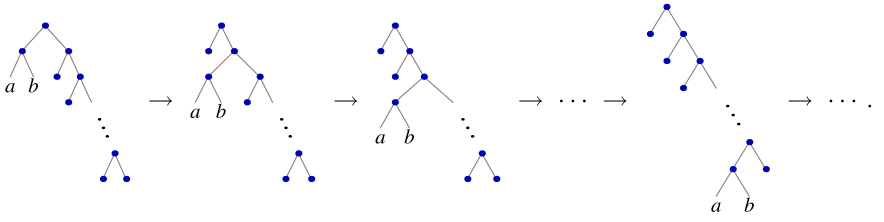
The image does not match t_3 at the root, so t_3 does not appear above the root of the new t_2.

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

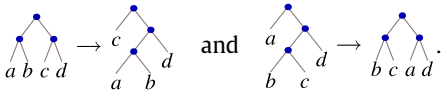
A natural question is whether for any two equivalent binary tree patterns s and t there exists a sequence of replacement bijections and left–right reflections that establishes their equivalence. For tree patterns of at most 7 leaves the answer is "Yes", which perhaps suggests that these maps suffice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

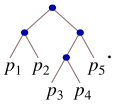
In this section we discuss qualitative results regarding this conjecture.

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.

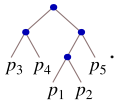
We return briefly to the replacement rule of Section 5.1 to mention that a minor modification produces a bijection on the full set of binary trees. Namely, take the two replacement rules



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:



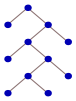
Now we check that applying each of the two replacement rules to this tree produces the same labeled tree, namely



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.

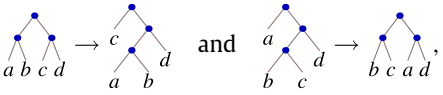
We conclude with a curious example in which two tree patterns can only be proven equivalent by a two-rule bijection that does not involve them directly. The trees

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

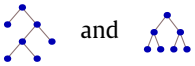
in class 6.5 are avoiding-equivalent by the permutation 126345, but neither

$t_{17} =$ 

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



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



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

Acknowledgments

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 A. 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 [8] using Singular via the interface package by Manuel Kauers and Viktor Levandovskyy [6]. 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).

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

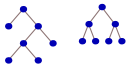
Class 3.1 (2 trees).

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

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).

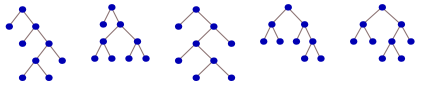


$$xyf^2 + \left(-2x^2(y-1) - 1\right)f + \left(x^3(y-1) + x\right) = 0$$

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).

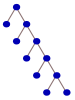


$$\left(xy - x^3(y-1)\right)f^2 + \left(x^2\left(x^2-2\right)(y-1) - 1\right)f + \left(x^3(y-1) + x\right) = 0$$

Class 5.3 (2 trees).

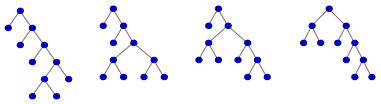
$$xyf^3 + \left(-3x^2(y-1) - 1\right)f^2 + \left(3x^3(y-1) + x\right)f - x^4(y-1) = 0$$

Class 6.1 (2 trees).



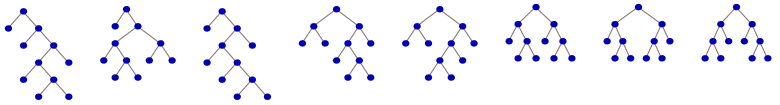
$$-x^5(y-1)f^4 - x^4(y-1)f^3 + \big(xy - x^3(y-1)\big)f^2 + \big(-x^2(y-1)-1\big)f + x = 0$$

Class 6.2 (8 trees).



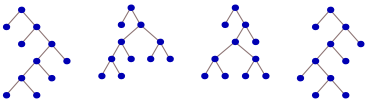
$$-x^4(y-1)f^3 + x\big(x^2\big(x^2-1\big)(y-1)+y\big)f^2 + \big(x^2\big(x^2-2\big)(y-1)-1\big)f + \big(x^3(y-1)+x\big) = 0$$

Class 6.3 (14 trees).



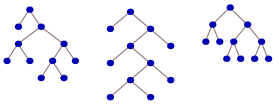
$$x\big(x^2\big(x^2-2\big)(y-1)+y\big)f^2 + \big(2x^2\big(x^2-1\big)(y-1)-1\big)f + \big(x^3(y-1)+x\big) = 0$$

Class 6.4 (8 trees).



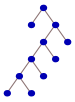
$$\big(xy - x^3(y-1)\big)f^3 + \big(x^2\big(2x^2-3\big)(y-1)-1\big)f^2 + \big(-x^5(y-1) + 3x^3(y-1) + x\big)f - x^4(y-1) = 0$$

Class 6.5 (6 trees).



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

Class 6.6 (2 trees).



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

Class 6.7 (2 trees).

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

References

[1] Emeric Deutsch, Dyck path enumeration, Discrete Math. 204 (1999) 167–202.
[2] Robert Donaghey, Louis Shapiro, Motzkin numbers, J. Combin. Theory Ser. A 23 (1977) 291–301.
[3] Philippe Flajolet, Robert Sedgewick, Analytic Combinatorics, Cambridge University Press, 2009.
[4] Philippe Flajolet, Paolo Sipala, Jean-Marc Steyaert, Analytic variations on the common subexpression problem, in: Automata, Languages, and Programming, in: Lecture Notes in Comput. Sci., vol. 443, 1990, pp. 220–234.
[5] Ian Goulden, David Jackson, An inversion theorem for cluster decompositions of sequences with distinguished subsequences, J. Lond. Math. Soc. (2) 20 (1979) 567–576.
[6] Manuel Kauers, Viktor Levandovskyy, Singular [a *Mathematica* package], available from http://www.risc.uni-linz.ac.at/research/combinat/software/Singular/index.html.
[7] John Noonan, Doron Zeilberger, The Goulden–Jackson cluster method: extensions, applications, and implementations, J. Difference Equ. Appl. 5 (1999) 355–377.
[8] Eric Rowland, TreePatterns [a *Mathematica* package], available from http://math.tulane.edu/~erowland/packages.html.
[9] Aris Sapounakis, Ioannis Tasoulas, Panos Tsikouras, Counting strings in Dyck paths, Discrete Math. 307 (2007) 2909–2924.
[10] Neil Sloane, The encyclopedia of integer sequences, http://www.research.att.com/~njas/sequences.
[11] Richard Stanley, Enumerative Combinatorics, vol. 2, Cambridge University Press, New York, 1999.
[12] Jean-Marc Steyaert, Philippe Flajolet, Patterns and pattern-matching in trees: an analysis, Inform. Control 58 (1983) 19–58.