DISSERTATION

Computer-Assisted Analytic Methods for Discrete Problems

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors der technischen Wissenschaften unter der Leitung von

Ao. Univ. Prof. Dipl.-Ing. Dr.techn. Michael Drmota,
104,
Institut für Diskrete Mathematik und Geometrie,

eingereicht an der Technischen Universität Wien,
Fakultät für Mathematik und Geoinformation,

von

Thomas Klausner,

Matr. Nr. 9325658,

Steinbachstraße 34-36, 3001 Mauerbach.

Wien, 17. Mai 2004 Thomas Klausner

Ph. D. Thesis

Computer-Assisted Analytic Methods for Discrete Problems

written under the supervision of
Ao. Univ. Prof. Dipl.-Ing. Dr. techn. Michael Drmota

at the Institute of Discrete Mathematics and Geometry (104),
Faculty of Mathematics and Geoinformation,
Vienna University of Technology, Austria,

by

Thomas Klausner,

9325658,

Steinbachstraße 34-36, 3001 Mauerbach, Austria.

Kurzfassung

Diese Doktorarbeit beschäftigt sich mit mathematischen Methoden, die es erlauben, mit Computer-Unterstützung diskrete mathematische Probleme asymptotisch zu behandeln. Konkret werden dabei zwei verschiedene Probleme behandelt.

Im ersten Teil der Arbeit werden Muster in (markierten) Bäumen untersucht. Ein Muster ist ein bestimmter vorgegebener Baum. Wir sagen, dass ein Muster auf einen bestimmten Teilbaum passt, wenn das Muster an dieser Stelle ein induzierter Teilbaum ist, das heißt, die Knotengrade der Knoten und ihre Nachbarschaftsrelationen im Baum mit denen der entsprechenden Knoten im Muster übereinstimmen. Einzelne Knoten können von mehreren Instanzen eines Musters verwendet werden, d.h. Überlappungen von Mustern im Baum sind erlaubt.

Wir nehmen alle Bäume einer bestimmten Größe n als gleich wahrscheinlich an und untersuchen, wie oft das vorgegebene Muster im Durchschnitt vorkommt. Dazu bedienen wir uns erzeugender Funktionen auf dieselbe Art, in der üblicherweise Bäume gezählt werden, führen aber eine zweite Variable ein, die die Muster mitzählt. Damit wir das effektiv machen können, untersuchen und zerlegen wir das Muster in einem ersten Schritt und erzeugen eine Art "Korrelationspolynom" (eigentlich ein System von Polynomen). Die dazu verwendeten Algorithmen werden ausführlich beschrieben.

Durch allgemeine Aussagen über diese Polynome können wir zeigen, dass die Anzahl der Muster einen zentralen Grenzwertsatz erfüllt. Im Speziellen kann für jedes Muster auch durch Lösung eines zugehörigen polynomiellen Gleichungssystems der Erwartungswert explizit berechnet werden. Dies wird anhand eines Beispiels vorgeführt. Alle beschriebenen Algorithmen lassen sich in MAPLE für beliebige Muster automatisch durchführen.

Wir besprechen weiters einige Möglichkeiten der Verallgemeinerung auf andere Muster oder andere Graphen.

Dieser Teil der Dissertation basiert auf der gemeinsamen Arbeit "The Distribution of Patterns in Random Trees" mit Frédéric Chyzak und Michael Drmota.

Im zweiten Teil der Arbeit beschäftigen wir uns mit dem Begriff der "zulässigen" (auch Hayman-zulässig bzw. Hayman-admissible genannten) Funktionen, den Hayman 1956 in [Hay56] eingeführt hat. Er bewies, dass geeignet normalisierte Koeffizienten von zulässigen Funktionen asymptotisch einer Normalver-

teilung folgen. Hayman präsentierte auch eine Liste von Basisfunktionen und Abschlussbedingungen, unter denen aus Hayman-zulässigen Funktionen andere Hayman-zulässige Funktionen erzeugt werden können. Seine Methode lässt sich jedoch leider nicht direkt zum Zählen kombinatorischer Objekte verwenden.

Wir verallgemeinern Haymans Ergebnis auf Funktionen in zwei Variablen, um damit einen weiteren Parameter untersuchen zu können. Das eine Ziel dabei ist es, eine kombinatorische Interpretation der Koeffizienten erzeugender Funktionen zuzulassen, das andere, ähnlich starke Abschlussbedingungen wie bei Hayman zur Verfügung zu stellen. Wir präsentieren diese Verallgemeinerung und ein MAPLE-Programm, das für eine beliebige gegebene Funktion automatisch testet, ob sie zulässig ("extended admissible") ist.

Eine kommentierte Version des MAPLE-Quelltextes ist Teil der Arbeit, und die darin verwendeten Algorithmen sind im Detail beschrieben.

Wir wenden das Konzept auch auf einige kombinatorische Beispiele an.

Dieser Teil der Dissertation basiert auf der gemeinsamen Arbeit "Extended Admissible Functions and Gaussian Limiting Distributions" mit Michael Drmota und Bernhard Gittenberger.

Abstract

The subject of this thesis is the application of computer algebra systems, MAPLE in particular, to asymptotic problems coming from discrete mathematics. Two specific problems are examined.

The first is concerned with patterns in labeled trees. We count the average number of times a particular pattern matches a tree of size n. Assuming that every tree of size n is equally likely, it is shown that the limiting distribution of the number of occurrences of the pattern is asymptotically normal, with mean value $\sim \mu n$ and variance $\sim \sigma^2 n$ with computable constants $\mu > 0$ and $\sigma \geq 0$. We provide an algorithm to compute μ explicitly and a MAPLE program that does most of the work. This part of the thesis is based on the paper "The Distribution of Patterns in Random Trees" coauthored with Frédéric Chyzak and Michael Drmota.

In the second part of the thesis, we generalize a class of functions which were introduced by Hayman and which we thus call Hayman-admissible. Hayman proved that the suitably normalized coefficients of these functions asymptotically follow a Gaussian distribution. He also assembled a useful list of closure properties, i.e., operations on Hayman-admissible functions that generate other Hayman-admissible functions. We generalize Hayman's result to functions in two dimensions, conserving many closure properties. We also present a MAPLE program that tests if a given function belongs to this class. This part of the thesis is based on the paper "Extended Admissible Functions and Gaussian Limiting Distributions" coauthored with Michael Drmota and Bernhard Gittenberger.

Contents

Chapter 1

Patterns in Trees

We take a particular (finite) tree $\mathcal{M}$ and and the set of unrooted labeled trees of size n $\mathcal{T}_n$ and count the number of occurrences of $\mathcal{M}$ as induced subtree in $\mathcal{T}_n$ asymptotically as $n \to \infty$.

Assuming that every tree of $\mathcal{T}_n$ is equally likely, it is shown that the limiting distribution of the number of occurrences of $\mathcal{M}$ as an induced subtree is asymptotically normal with mean value $\sim \mu n$ and variance $\sim \sigma^2 n$ with computable constants $\mu > 0$ and $\sigma \geq 0$.

An algorithm is given that provides a system of equations, and its implementation in MAPLE is presented. Solving this system, μ can be computed exactly.

1.1 Introduction

Pattern matching is currently a growing field, enjoying attention of researchers from such diverse fields as molecular biology, information retrieval, pattern recognition, compiling, and many other areas (see for example [Lon04]). Usually, the task is to find combinatorial properties that enable one to find patterns more quickly.

Pattern matching is done on various different structures, starting from words to more complex data structures like trees and graphs or regular expressions.

In particular, we restrict ourselves to labeled trees. Trees are connected graphs without cycles. They are used in many areas, especially in computer science; for example, as data structures, in searching algorithms, for storing data, and for optimizing expressions for compilers (see e.g. [ASU86]).

Also, we do not concern ourselves with algorithms for pattern matching, but rather try to get asymptotic results on the number of occurrences of a pattern in trees. In particular, we count the occurrences of a pattern in all labeled trees of size n. We then normalize the distributions of these counts by subtracting their mean and dividing by the square root of the variance. The resulting sequence (for varying n) of distributions weakly converges to the standard normal distribution

for $n \to \infty$, in other words, it follows a central limit theorem.

Some work has already been done on similar topics. The distribution of *stars* (nodes of given degree) has been discussed in [DG99] for various classes of trees. Some previous work for unlabeled trees is due to Robinson and Schwenk [RS75]. Patterns in (rooted) trees have also been investigated by Dershowitz and Zaks [DZ89]. However, they only consider patterns starting at the root. There is also some work on patterns in random binary search trees by Flajolet, Gourdon, and Martínez [FGM97]. They, too, obtain a central limit theorem. Flajolet and Steyaert also analyzed algorithms for pattern matchings in trees [FS80a, FS80b, SF83]. Further Ruciński [Ruc88] established conditions for when the number of occurrences of a given subgraph in random graphs follow a normal distribution.

Our aim was to prove the limit theorem stated above and to provide algorithms and tools in MAPLE to help compute the mean value for any given pattern automatically.

1.2 Definitions

We will concern ourselves with the following classes of trees:

Definition 1. A *tree* (or *unlabeled tree*) of size n is a connected graph with n nodes without cycles, i.e., there is exactly one path from one node of the tree to any other node of the tree. A *labeled tree* of size n is a tree where each node has been assigned a unique number from 1 to n. A (labeled or unlabeled) rooted tree is a (labeled or unlabeled) tree where one node has been designated as *root*. A (labeled or unlabeled) *planted tree*, or *planted rooted tree*, is a (labeled or unlabeled) rooted tree where the root has been adjoined an additional node which is not counted (in effect increasing the root node's degree by one).

In the following we will only regard labeled graphs if not specified otherwise. A very useful construct are generating functions.

Definition 2. The *ordinary generating function* or just *generating function* of a sequence $a_n = a_0, a_1, a_2, \ldots$ is the formal power series $A(x) = \sum_{n=0}^{\infty} a_n x^n$.

For combinatorial objects, a common approach is to let the n-th element a_n of the sequence count the objects of size n; in other words, the coefficient a_n of x^n in $A(x)$, in symbols $[x^n]A(x)$, denotes the number of objects of size n.

There is a wide range of combinatorial constructions that can be directly translated into operations on generating functions. For labeled constructs, exponential generating functions have better properties.

Definition 3. The *exponential generating function* of a sequence a_n is the formal power series $A(x) = \sum_{n=0}^{\infty} a_n \frac{x^n}{n!}$.

Usually, combinatorial constructions are converted to operations on the generating functions, then the generating functions are computed at least so far that the coefficients in which one is interested can be read off again. These coefficients then provide object counts.

For a detailed introduction into generating functions and combinatorial constructions refer to [FS02].

We will mostly be using two simple equivalences between combinatorial constructs and the corresponding generating functions which hold for both ordinary and exponential generating functions:

- Union: $A = B + C \Longrightarrow A(x) = B(x) + C(x)$
- Product: $A = B \times C \Longrightarrow A(x) = B(x) \cdot C(x)$

1.2.1 Counting Stars in Trees

We follow a three-step program to count the number of trees in the class of all labeled trees of size n, $\mathcal{T}_n$, and then use the same program to count the number of occurrences of nodes of degree k in $\mathcal{T}_n$.

For this purpose we make use of the sets $\mathcal{R}_n$ of rooted labeled trees of size n and $\mathcal{P}_n$ of planted labeled trees of size n.

Obviously $|\mathcal{P}_n| = |\mathcal{R}_n|$, since adding an edge in each tree doesn't change the number of nodes. Also, $|\mathcal{T}_n| = |\mathcal{R}_n|/n$, since each unrooted tree can be rooted at each of its n nodes, and we get a different rooted tree because we are only regarding labeled trees. It is also well known that $|\mathcal{R}_n| = n^{n-1}$ and $|\mathcal{T}_n| = n^{n-2}$, see for example [HP73].

The three steps of the program are:

1. Determine the number of planted trees $\mathcal{P}_n$.
2. Use $\mathcal{P}_n$ to count rooted trees $\mathcal{R}_n$.
3. Compute the number of unrooted trees $\mathcal{T}_n$ from $\mathcal{R}_n$ or $\mathcal{R}_n$ and $\mathcal{P}_n$.

We define three exponential generating functions

$$p(x) = \sum_{n=0}^{\infty} |\mathcal{P}_n| \frac{x^n}{n!}, \quad r(x) = \sum_{n=0}^{\infty} |\mathcal{R}_n| \frac{x^n}{n!}, \quad t(x) = \sum_{n=0}^{\infty} |\mathcal{T}_n| \frac{x^n}{n!}$$

and proceed in the following way:

1. **Planted Rooted Trees:** A planted tree is a (planted) root node with zero, one, two, ... planted subtrees. The order of the subtrees does not matter.

$$P = \circ \cup (\circ \times P) \cup (\circ \times P \times P) \cup (\circ \times P \times P \times P) \cup \ldots$$

Converting this construction to a generating function yields:

$$p(x) = \sum_{n=0}^{\infty} \frac{xp(x)^n}{n!} = xe^{p(x)}.$$

2. **Rooted Trees:** For rooted trees we get the same result, despite of the planted root which is not present here: just a root with zero, one, two, ... planted subtrees.

$$R = \circ \cup (\circ \times P) \cup (\circ \times P \times P) \cup (\circ \times P \times P \times P) \cup \ldots$$

This gives

$$r(x) = \sum_{n=0}^{\infty} \frac{xp(x)^n}{n!} = xe^{p(x)} = p(x).$$

3. **Unrooted Trees:** Finally, we have $|\mathcal{T}_n| = |\mathcal{R}_n|/n$ (as already mentioned). However, we can also express $t(x)$ in the following way:

Lemma 1.

$$t(x) = r(x) - \frac{1}{2}p(x)^2.$$

Proof. This relation follows from a natural bijection between rooted trees on the one hand and unrooted trees and pairs of planted rooted trees on the other hand.

The bijection is as follows: Consider the class of rooted (labeled) trees. If the root is labeled by 1 then consider the tree as an unrooted tree. If the root is not labeled by 1 then take the first edge of the path between the root and 1 and cut the tree into two planted rooted tree at this edge.

The other direction of the bijection: If you have a single unrooted tree, root it at the node labeled with 1. If you have two planted rooted trees, connect their planted edges and root the tree at the planted root of that tree that does not contain the node labeled with 1. □

The functional equation for $p(x)$ can either be used to extract the explicit number $|\mathcal{P}_n| = n^{n-1}$ via Lagrange inversion (e.g., see [HP73]) or to obtain the radius of convergence and asymptotic expansions of the singular behavior of this function. By direct application of the implicit function theorem, one can compute that $x_0 = 1/e$ is the common radius of convergence of $p(x)$, $r(x)$, and $t(x)$. By subsequent application of iteration methods ("bootstrapping") (e.g., see [dB81]), one finds the singularity at $x = x_0$ is of square-root type:

$$p(x) = r(x) = 1 - \sqrt{2}\sqrt{1 - ex} + \frac{2}{3}(1 - ex) + \cdots,$$

$$t(x) = \sqrt{e} + \frac{2\sqrt{2e}}{3}(1 - ex)^{3/2} + \cdots$$

which is reflected by the asymptotic expansions of the numbers

$$|\mathcal{P}_n| = |\mathcal{R}_n| = n^{n-1} \sim \frac{n!}{\sqrt{2\pi}} e^n n^{-3/2},$$
$$|\mathcal{T}_n| = n^{n-2} \sim \frac{n!}{\sqrt{2\pi}} e^n n^{-5/2}.$$

In order to demonstrate the usefulness of the three-step procedure above (which on the first glance might seem to be redundant) we repeat the same steps for counting *stars* with k edges in trees, resp. the number of nodes of degree k (where $k \geq 1$ is supposed to be a given fixed number). We introduce a second variable, u, to keep count of the number of stars. Let $p_{n,m}$ denote the number of planted trees of size n with exactly m nodes of degree k. Furthermore, let $r_{n,m}$ and $t_{n,m}$ be the corresponding numbers for rooted and unrooted trees and set

$$p(x,u) = \sum_{n,m=0}^{\infty} p_{n,m} \frac{x^n u^m}{n!},$$
$$r(x,u) = \sum_{n,m=0}^{\infty} r_{n,m} \frac{x^n u^m}{n!},$$
$$t(x,u) = \sum_{n,m=0}^{\infty} t_{n,m} \frac{x^n u^m}{n!}.$$

As above, we use the recursive construction of planted rooted trees as a root with 0, 1, ... planted rooted trees attached. However, for counting stars with k nodes, we mark roots with exactly k nodes with u, and arrive at

$$p(x,u) = \sum_{\substack{n=0 \\ n \neq k-1}}^{\infty} \frac{xp(x,u)^n}{n!} + \frac{xup(x,u)^{k-1}}{(k-1)!} = xe^{p(x,u)} + \frac{x(u-1)p(x,u)^{k-1}}{(k-1)!}.$$

Similarly, we get for rooted trees:

$$r(x,u) = \sum_{\substack{n=0 \\ n \neq k}}^{\infty} \frac{xp(x,u)^n}{n!} + \frac{xup(x,u)^k}{k!} = xe^{p(x,u)} + \frac{x(u-1)p(x,u)^k}{k!}.$$

For unrooted trees, we could choose the easy way out and take $t_{n,m} = r_{n,m}/n$, which is sufficient for our purposes. However, (as above) it is also possible to express $t(x,u)$ (as above) in the following way:

$$t(x,u) = r(x,u) - \frac{1}{2}p(x,u)^2.$$

Note that the use of the notion of planted trees is crucial in order to keep track of the nodes of degree k.

In [DG99] this approach was used to show that the asymptotic distribution of the number of nodes of degree k in trees of size n is normal, with expectation and variance proportional to n.

1.3 Counting Patterns in Trees

1.3.1 Patterns and Pattern Matching

We will now define patterns and generalize the counting procedure from the previous section to patterns.

Definition 4. A *pattern* is an unrooted unlabeled tree. A *planted pattern* is a planted rooted unlabeled tree.

Definition 5. A planted pattern $\mathcal{M}_p$ *matches* a planted rooted tree $\mathcal{T}_n$ if $\mathcal{M}_p$ occurs as an induced subtree starting at the planted root, that is, the branch structure and node degrees (except for leaves in $\mathcal{M}_p$) match.

For example, in figure 1.1 the planted pattern matches the planted tree a, but not b.

In the more general case, we could mark up nodes in the pattern $\mathcal{M}_p$ as *filled* and *empty* and only demand that the degree of filled nodes matches the degree of the corresponding node in $\mathcal{T}_n$, while unfilled nodes in $\mathcal{M}_p$ match nodes of any degree in $\mathcal{T}_n$. Depending on the resulting structure, this may make the problem quite a bit more difficult; see section 1.5 for some examples. The default case corresponds to the general case where all internal nodes are filled and the leaves are empty. For stars, $\mathcal{M}$ consists of a (filled) node with exactly k (empty) connected nodes.

In the following, we will take the graph from figure 1.2 as example pattern $\mathcal{M}$.

Our basic idea for counting patterns is to create all trees of size n by creating them recursively from the bottom up and counting patterns when they first appear. To do so, we have to consider all possibilities how they might appear.

The first step is to create all planted patterns from a pattern. We get all planted patterns for a pattern by taking each leaf in turn and planting the pattern on the edge from the leaf to the connected internal node. Multiple leaves connecting to the same internal node only give one new planted pattern. If any of the planted patterns we construct in this way matches at the root during the recursive creation of all trees, the corresponding tree contains a new pattern occurrence we need to count.

For the example pattern, we arrive at three planted patterns, see figure 1.3.

The next step follows from noticing that these planted patterns usually have heights greater than 1, while the recursion creating the trees (attaching $0, 1, \ldots$ nodes to a node) is only one level deep. We adapt to this by splitting the pattern into subpatterns, each of height one, and matching those instead.

Another problem is that some of those subpatterns occur in multiple places, so we have to identify and handle these cases appropriately. And of course, we want all of this to be done automatically.

Let us begin by formulating the proposition describing the generating functions for the subpatterns and their relation with the generating function of trees.

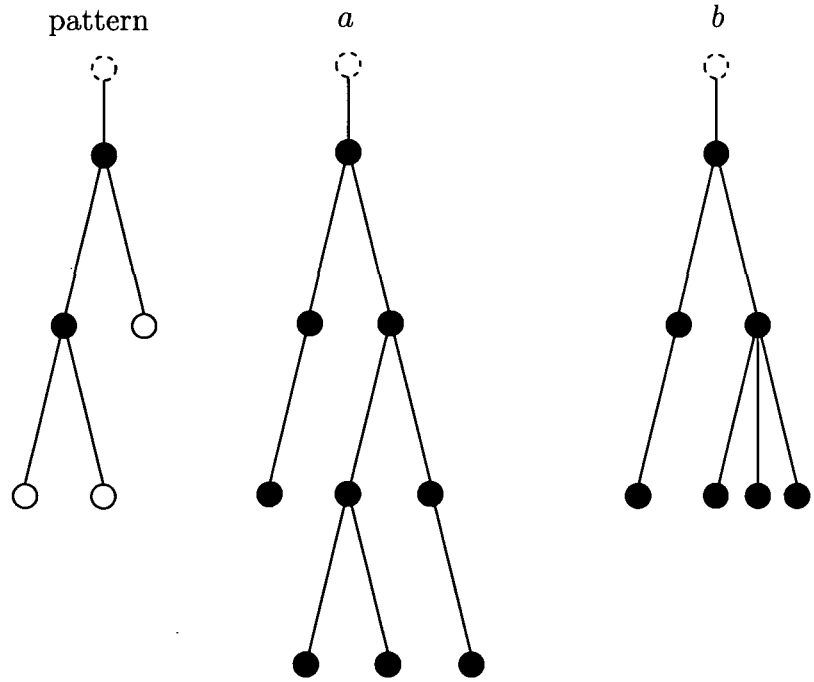

Figure 1.1: Planted pattern matching: a matches the pattern, b does not

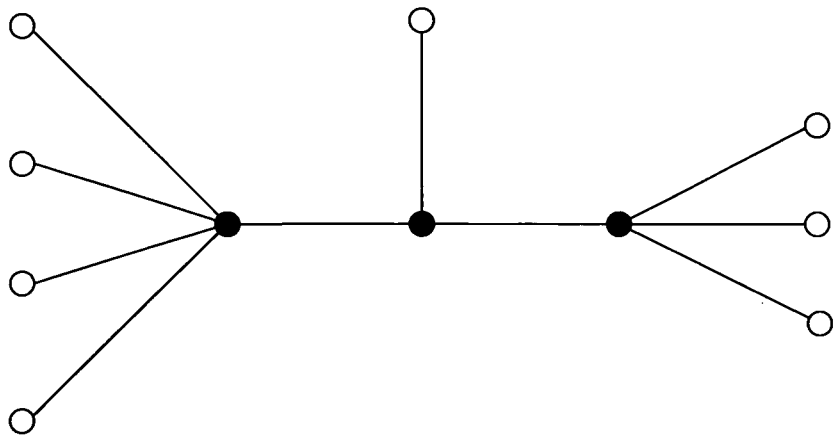

Figure 1.2: Example pattern

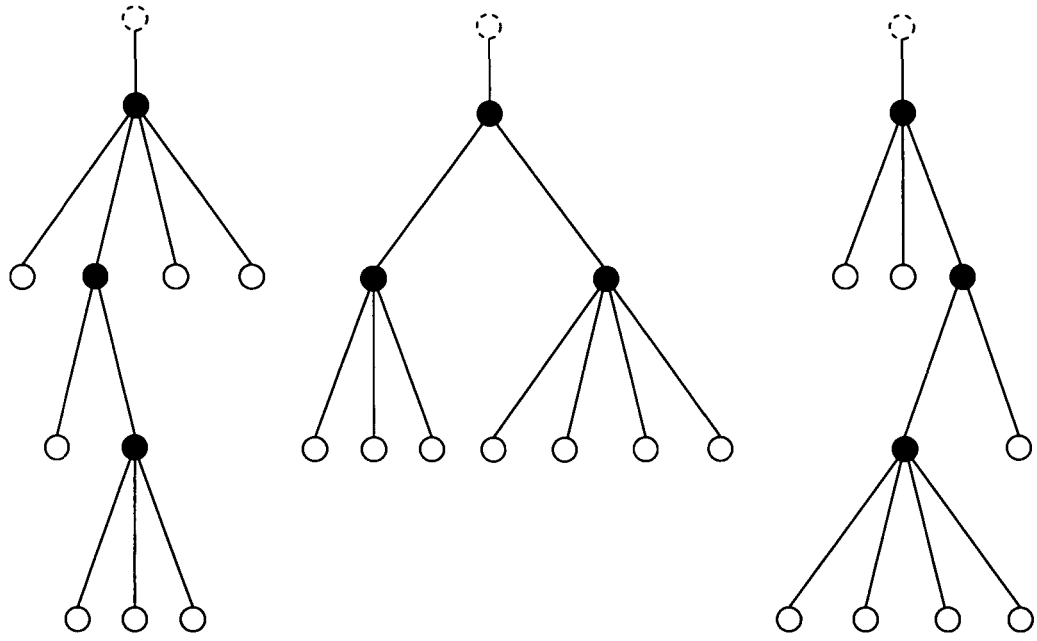

Figure 1.3: Planted patterns for graph from figure 1.2

Let $p_{n,m}$ denote the number of planted rooted trees with n nodes and m occurrences of the pattern $\mathcal{M}$ and

$$p = p(x,u) = \sum_{n,m=0}^{\infty} p_{n,m} \frac{x^n u^m}{n!}$$

the corresponding generating function.

We also need the following definition of a dependency graph for a system of equations:

Definition 6. For a system of equations $a_i = Q_i(a_1, \ldots, a_n), i = 1, \ldots, n$, where each $Q_i(a_1, \ldots, a_n)$ is a formal power series in a_i, we define the following graph: Create one node for each a_i. Draw a directed edge from a_i to a_j if and only if

$$\exists n \geq 1 : [a_j^n] Q_i(a_1, \ldots, a_n) \neq 0.$$

We call this graph the *dependency (di)graph* of the system of equations.

Proposition 1. *(***Planted Rooted Trees***) Let $\mathcal{M}$ be a pattern. Then there exist a certain number L of auxiliary functions $a_j(x,u)$ $(0 \leq j \leq L-1)$ with*

$$p(x,u) = \sum_{j=0}^{L-1} a_j(x,u)$$

and polynomials $P_j(y_1, \ldots, y_L, u)$ $(1 \leq j \leq L-1)$ *with non-negative coefficients such that*

$$\begin{aligned} a_0(x,u) &= xe^{a_0(x,u)+\cdots+a_{L-1}(x,u)} - x\sum_{j=1}^{L-1} P_j(a_0(x,u),\ldots,a_{L-1}(x,u),1) \\ a_1(x,u) &= x \cdot P_1(a_0(x,u),\ldots,a_{L-1}(x,u),u) \\ &\vdots \\ a_{L-1}(x,u) &= x \cdot P_{L-1}(a_0(x,u),\ldots,a_{L-1}(x,u),u). \end{aligned} \tag{1.1}$$

Furthermore,

$$\sum_{j=1}^{L-1} P_j(y_0,\ldots,y_{L-1},1) \leq_c e^{y_0+\cdots+y_{L-1}},$$

where $f \leq_c g$ *means that all Taylor coefficients of the left hand side are smaller than or equal to the corresponding coefficients of the right hand side.*

Moreover, the dependency graph of this system is strongly connected, that is, there is a path from every node to every other node.

We will now describe algorithms to convert a given pattern to a system of polynomial equations of this type.

1.3.2 Conversion Algorithms

The goal of the algorithms in this subsection is to derive a description for the forest of the planted subtrees induced by a given pattern in terms of disjoint unions of classes of subtrees. These classes of subtrees themselves are recursively defined as unions and Cartesian products of these classes. The resulting recursions then describe the autocorrelation of the pattern. The original non-disjoint subtrees can be expressed as unions of the disjoint classes of subtrees, and thus we can use our knowledge about the disjoint classes to solve the original problem. We proceed in two steps and introduce two algorithms below:

1. We convert a given pattern into a recursive description of the forest of its planted subtrees. This calculation is reminiscent of the DAGification process of computer science (see for example [ASU86]), which aims at compacting an expression tree by sharing repeated subexpressions. This conversion results in recursive equations between the classes corresponding to the planted subtrees of the original pattern. However, this description is still ambiguous: the intersection of two classes need not be empty.

2. We disambiguate the recursive description by determining a partition of the class p of all trees that is better adapted to the family of classes for our pattern. In other words, we split the non-disjoint classes into smaller ones,

so that no two new classes have a non-empty intersection; meanwhile, we update the recursive descriptions of the original patterns in terms of these disjoint classes.

In view of the algebraic nature of the recursive equations for classes of trees—one could even speak of their "polynomial" nature—we introduce the notation $\bigoplus_{i\in I} t_i$ for the disjoint union of classes t_i, $\bigcup_{i\in I} t_i$ for the not necessarily disjoint union of classes, and $\bigotimes_{i=1}^{r} t_i$ to mean the Cartesian product of classes t_i. All three operations are commutative, $\otimes$ binds stronger than $\oplus$ or $\cup$, and $\otimes$ is also distributive over $\oplus$ and $\cup$. The products of classes of trees t_i are thus, in a way, commutative "monomials".

Additionally, let λ to denote a generalized integer partition in the sense that a part may be zero. The notation $\ell(\lambda)$ denotes the length of a generalized partition (including any zeroes).

DAGification of the Forest of Planted Subtrees of a Pattern

The DAGification algorithm needs to remember the sets of classes associated with the children of each node it has dealt with, as well as the class that has been assigned to earlier occurrences of such children classes. The way to do so is by operating an associative table that assigns a class name t_i to products of the form $\bigotimes_{j=1}^{\ell(\lambda)} t_{\lambda_j}$. This table is called the "uniquification table" in the following algorithm description.

Note that we give a description of the algorithm that takes the graph structure of the pattern as its input, but we could equivalently start with some representation of any rooted covering tree of the pattern, from which we could first compute the adjacency lists.

Input: graph structure of the pattern, given by adjacency lists

Output: a minimal recursive description of the forest of planted subtrees of the pattern, of the form

$$t_i = \bigotimes_{j=1}^{\ell(\lambda^{(i)})} t_{\lambda_j^{(i)}} \qquad \text{for } 1 \le i \le m$$

with the constraint $\lambda_j^{(i)} < i$ for all i and j

Algorithm:

(Initialization) Introduce the exceptional class t_0 to denote the planted tree consisting of a single node (in other words, a leaf)

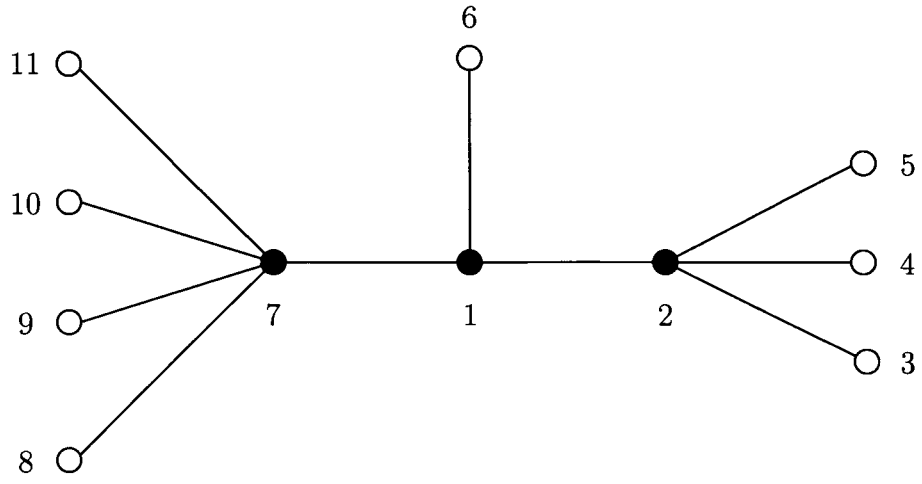

Figure 1.4: Labeled example pattern

(Main loop) For all leaves ℓ of the pattern, that is, for all nodes whose adjacency list consists of a single element, perform a depth-first traversal of the tree, starting from ℓ; during this recursive calculation, at each node n:

1. if the node is a leaf, return the class t_0
2. else, recursively determine the class associated with each child of n
3. write the subtree rooted at n as a (commutative) product $\pi = t_{\lambda_i} \otimes \cdots \otimes t_{\lambda_r}$ of the classes obtained in the previous step
4. look this product up in the uniquification table to check whether it has already been assigned a class t_i
5. if not, create a new class t_i, remember its definition $t_i = \pi$, and assign t_i to the product π in the uniquification table
6. return the class t_i, whether it was just created or found by lookup

(Conclusion) Return the sequence of definitions of the form $t_i = \pi$, for $i = 1, 2, \ldots$

This algorithm only identifies absolutely identical subtrees, it does not guarantee that any two resulting t_i and t_j have an empty intersection (and in general, there will be non-empty intersections).

Starting from the example pattern of figure 1.2, we label the nodes from 1 to 11 as in figure 1.4. This numbering corresponds to a depth-first traversal starting from the central node 1, which we arbitrarily chose as starting point, dealing with its neighbors from right to left. As a consequence, the implicit rooting of

the pattern is from the central-most node (Case b in figure 1.6). We thus have the adjacency lists

$$1 \mapsto \{2,6,7\}, \quad 2 \mapsto \{1,3,4,5\}, \quad 3 \mapsto \{2\}, \quad 4 \mapsto \{2\}, \quad 5 \mapsto \{2\}, \quad 6 \mapsto \{1\},$$
$$7 \mapsto \{1,8,9,10,11\}, \quad 8 \mapsto \{7\}, \quad 9 \mapsto \{7\}, \quad 10 \mapsto \{7\}, \quad 11 \mapsto \{7\},$$

where a node is mapped to its set of neighbors. We make the assumption that leaves and adjacency lists are processed in increasing order of the node numbering.

We iterate over all leaves. The first leaf to consider is 3. By a depth-first traversal from it, the bottom-up construction of planted subtrees first considers the subtree planted at 8, corresponding to the special class t_0; the subtrees planted at 9, 10, and 11 are recognized to be the same class. Next, a class $t_1 = t_0^{\otimes 4}$ is introduced to correspond to the subtree planted at 7, followed by a class $t_2 = t_1 \otimes t_0$ for the subtree planted at 1, and by a class $t_3 = t_2 \otimes t_0^{\otimes 2}$ for the subtree planted at 2.

The next leaves, 4 and 5, introduce no additional classes and return the same class t_3 as leaf 3.

Dealing with leaf 6, the depth-first traversal first considers the branch planted at 2, and introduces a class $t_4 = t_0^{\otimes 3}$ for the subtree planted at 2, then a class $t_5 = t_4 \otimes t_1$ for the subtree planted at 1, after realizing that the branch planted at 7 has already been considered and assigned the class t_1.

The last leaf to introduce new classes is 8, since leaves 9, 10, and 11 correspond to isomorphic planted structures: the subtree planted at 2 has already been considered and returns the class t_4; the subtree planted at 1 now gives rise to a class $t_6 = t_4 \otimes t_0$; the subtree planted at 7 finally introduces a class $t_7 = t_6 \otimes t_0^{\otimes 3}$.

Summing up, the algorithm returns:

$$t_1 = t_0^{\otimes 4}, \qquad t_2 = t_1 \otimes t_0, \qquad t_3 = t_2 \otimes t_0^{\otimes 2}, \qquad t_4 = t_0^{\otimes 3},$$
$$t_5 = t_4 \otimes t_1, \qquad t_6 = t_4 \otimes t_0, \qquad t_7 = t_6 \otimes t_0^{\otimes 3}.$$

Disambiguating the Autocorrelation of a Tree Pattern

The idea of the algorithm below is to consider each class t_i of trees in turn, introducing its defining equation

$$t_i = \bigotimes_{j=1}^{\ell(\lambda^{(i)})} t_{\lambda_j^{(i)}}$$

into the calculation, while maintaining (and refining) a partition

$$p = \bigoplus_{i=0}^{n} a_i$$

of the total class of plane trees. To be able to do so, it is crucial that the recursive equation for t_i refers to classes t_j with $j < i$ only, starting with the special class $t_0 = p$, the full class of patterns.

At any stage in the algorithm, the class of r-ary trees is given as the disjoint union of Cartesian products

$$\bigoplus_{\lambda \in \Lambda} \bigotimes_{j=1}^{r} a_{\lambda_j} \qquad \text{where} \qquad \Lambda = \{ \lambda : \ell(\lambda) = r,\ 0 \leq \lambda_j \leq n \}.$$

In the process of the algorithm below, each class t_i gets represented in a "polynomial" form like above, indexed by a subset Λ of the set of partitions of a given length. Computing intersections and differences of classes means merely computing intersections and differences of the Λ in their representations.

In this setting, direct sums and tensor products behave like algebraic operations in a polynomial setting: in particular, a tensor product of direct sums expands by distributivity as a direct sum of direct products.

Input:

- recursive descriptions of classes of trees of the form

$$t_i = \bigotimes_{j=1}^{\ell(\lambda^{(i)})} t_{\lambda_j^{(i)}} \qquad \text{for } 1 \leq i \leq m$$

  with the constraint $\lambda_j^{(i)} < i$ for all i and j

Output:

- an integer n implying a partition

$$p = \bigoplus_{i=0}^{n} a_i$$

- a representation of each t_i of the form

$$t_i = \bigoplus_{j \in I_i} a_j \qquad \text{for } 0 \leq i \leq m \text{ and } I_i \subset \{0, \ldots, n\}$$

- a recursive description of the a_i of the form

$$a_i = \bigoplus_{\lambda \in \Lambda_i} \bigotimes_{j=1}^{\ell(\lambda)} a_{\lambda_j} \qquad \text{for } 1 \leq i \leq n,$$

  a_0 being implicitly described as $p \setminus \bigoplus_{i=1}^{n} a_i$

Algorithm:

(Initialization) Start with the trivial partition $p = a_0$ for $n = 0$, the single representation $t_0 = a_0$, that is, $I_0 = \{0\}$, and the trivial recursion $a_0 = a_0$, that is, $\Lambda_0 = \{(0)\}$

(Main loop) For k from 1 to m do

1. replace each t_i in the definition of t_k with its current representation in terms of the a_j, expand, and set s to the result, so as to get a representation of t_k of the form

$$s = \bigoplus_{\lambda \in \Lambda^{(s)}} \bigotimes_{j=1}^{\ell(\lambda)} a_{\lambda_j} \qquad \text{for some } \Lambda^{(s)}$$

2. for i from 1 to n while $s \neq \emptyset$ do
   (a) set b to $a_i \cap s$ and $\Lambda_\cap$ to $\Lambda_i \cap \Lambda^{(s)}$
   (b) if $b \neq \emptyset$, then do
      i. set b' to $a_i \setminus s$
      ii. if $b' \neq \emptyset$, then
         A. create a new a_j with description b': increment n before setting a_n to b', that is, before setting Λ_n to $\Lambda_i \setminus \Lambda^{(s)}$
         B. split a_i into $a_i \oplus a_n$ in the descriptions of the a_j, that is, add n into each set I_j containing i
         C. split a_i into $a_i \oplus a_n$ in the representations of the t_j, b, and s, that is, for each partition in each of the Λ_j, $\Lambda_\cap$, and $\Lambda^{(s)}$, add n when the partition involves i
         D. set a_i to b by setting Λ_i to $\Lambda_\cap$
      iii. set s to $s \setminus b$, which is also $s \setminus a_i$, and update $\Lambda^{(s)}$ by setting it to $\Lambda^{(s)} \setminus \Lambda_i$
3. if $s \neq \emptyset$, then
   (a) create a new a_j with description s: increment n before setting a_n to s, that is, before setting Λ_n to $\Lambda^{(s)}$
   (b) split a_0 into $a_0 \oplus a_n$ in the descriptions of the a_j, that is, add n into each set I_j containing 0
   (c) split a_0 into $a_0 \oplus a_n$ in the representations of the t_j, that is, for each partition in each of the Λ_j, add n when the partition involves 0
4. represent t_k as the union of all those a_is that have contributed a non-empty b at step (2b) and of a_n if a new a_j was created at step (3a), that is, create the corresponding set I_k consisting of the contributing is, together with n if relevant

(Final step) Return n, the representations of the t_i for $1 \leq i \leq m$, the descriptions of the a_i for $1 \leq i \leq n$

For example, running the algorithm with input from the example of the previous section goes through the following stages (where we keep expressions in factored form):

$k = 1$: from $t_1 = a_0^{\otimes 4}$, we derive $t_1 = a_1$ for $a_1 = (a_1 \oplus a_0)^{\otimes 4}$.

$k = 2$: from $t_2 = a_1 \otimes (a_1 \oplus a_0)$, we derive $t_1 = a_1$, $t_2 = a_2$ for $a_1 = p^{\otimes 4}$, $a_2 = a_1 \otimes p$, where $p = a_2 \oplus a_1 \oplus a_0$.

$k = 3$: from $t_3 = a_2 \otimes (a_2 \oplus a_1 \oplus a_0)^{\otimes 2}$, we derive $t_1 = a_1$, $t_2 = a_2$, $t_3 = a_3$ for $a_1 = p^{\otimes 4}$, $a_2 = a_1 \otimes p$, $a_3 = a_2 \otimes p^{\otimes 2}$, where $p = a_3 \oplus a_2 \oplus a_1 \oplus a_0$.

$k = 4$: from $t_4 = (a_3 \oplus a_2 \oplus a_1 \oplus a_0)^{\otimes 3}$, we derive $t_1 = a_1$, $t_2 = a_2$, $t_3 = a_3$, $t_4 = a_4 \oplus a_3$ for $a_1 = p^{\otimes 4}$, $a_2 = a_1 \otimes p$, $a_3 = a_2 \otimes p^{\otimes 2}$, $a_4 = (a_4 \oplus a_3 \oplus a_1 \oplus a_0)^{\otimes 3}$, where $p = a_4 \oplus a_3 \oplus a_2 \oplus a_1 \oplus a_0$.

$k = 5$: from $t_5 = (a_4 \oplus a_3) \otimes a_1$, we derive $t_1 = a_1$, $t_2 = a_5 \oplus a_2$, $t_3 = a_3$, $t_4 = a_4 \oplus a_3$, $t_5 = a_2$ for $a_1 = p^{\otimes 4}$, $a_2 = (a_4 \oplus a_3) \otimes a_1$, $a_3 = (a_5 \oplus a_2) \otimes p^{\otimes 2}$, $a_4 = (a_4 \oplus a_3 \oplus a_1 \oplus a_0)^{\otimes 3}$, $a_5 = (a_5 \oplus a_2 \oplus a_1 \oplus a_0) \otimes a_1$, where $p = a_5 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_1 \oplus a_0$.

$k = 6$: from $t_6 = (a_4 \oplus a_3) \otimes (a_5 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_1 \oplus a_0)$, we derive $t_1 = a_1$, $t_2 = a_5 \oplus a_2$, $t_3 = a_3$, $t_4 = a_4 \oplus a_3$, $t_5 = a_2$, $t_6 = a_6 \oplus a_2$ for $a_1 = p^{\otimes 4}$, $a_2 = (a_4 \oplus a_3) \otimes a_1$, $a_3 = (a_5 \oplus a_2) \otimes p^{\otimes 2}$, $a_4 = (a_6 \oplus a_4 \oplus a_3 \oplus a_1 \oplus a_0)^{\otimes 3}$, $a_5 = (a_6 \oplus a_5 \oplus a_2 \oplus a_1 \oplus a_0) \otimes a_1$, $a_6 = (a_6 \oplus a_5 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_0) \otimes (a_4 \oplus a_3)$, where $p = a_6 \oplus a_5 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_1 \oplus a_0$.

$k = 7$: from $t_7 = (a_6 \oplus a_2) \otimes (a_6 \oplus a_5 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_1 \oplus a_0)^{\otimes 3}$, we derive $t_1 = a_7 \oplus a_1$, $t_2 = a_5 \oplus a_2$, $t_3 = a_3$, $t_4 = a_4 \oplus a_3$, $t_5 = a_2$, $t_6 = a_6 \oplus a_2$, $t_7 = a_1$ for $a_1 = (a_6 \oplus a_2) \otimes p^{\otimes 3}$, $a_2 = (a_4 \oplus a_3) \otimes (a_7 \oplus a_1)$, $a_3 = (a_5 \oplus a_2) \otimes p^{\otimes 2}$, $a_4 = (a_7 \oplus a_6 \oplus a_4 \oplus a_3 \oplus a_1 \oplus a_0)^{\otimes 3}$, $a_5 = (a_7 \oplus a_6 \oplus a_5 \oplus a_2 \oplus a_1 \oplus a_0) \otimes (a_7 \oplus a_1)$, $a_6 = (a_6 \oplus a_5 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_0) \otimes (a_4 \oplus a_3)$, $a_7 = (a_7 \oplus a_5 \oplus a_4 \oplus a_3 \oplus a_1 \oplus a_0)^{\otimes 4}$, where $p = a_7 \oplus a_6 \oplus a_5 \oplus a_4 \oplus a_3 \oplus a_2 \oplus a_1 \oplus a_0$.

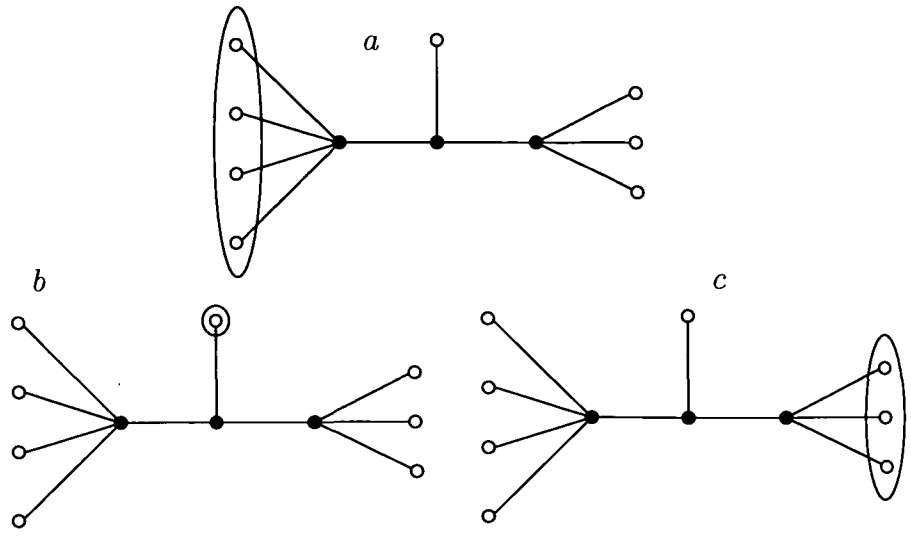

Figure 1.5: Possible planted roots in the pattern

Creating the Corresponding Polynomials

Let $I_j(l_0, \ldots, l_{L-1})$ be the indicator function of a term $\bigotimes_{i=0}^{L-1} a_i^{l_i}$ in a_j, that is, let $I_j(l_0, \ldots, l_{L-1}) = 1$ if there is a term $\bigotimes_{i=0}^{L-1} a_i^{l_i}$ in a_j and $I_j(l_0, \ldots, l_{L-1}) = 0$ otherwise.

Further, let $k(d, l_0, \ldots, l_{L-1})$ denote the number of occurrences of the pattern $\mathcal{M}$ at a node of degree d with l_i subtrees of type a_i.

We compute $k(d, l_0, \ldots, l_{L-1})$ iteratively. The combinations introducing a new occurrence of the pattern are identified by looking at the original pattern and rooting it in all nodes next to leaves. We do not have to root in nodes that have no neighboring leaves, because they cannot cause new patterns (there is no place for the additional edge to the planted root). We identify the subtrees that have to meet to form this pattern and express those subtrees with the help of the a_i from above in the following form:

- If all of $a_{i_1}, \ldots, a_{i_n}$ are necessary (duplicate indices allowed) for one occurrence, we write $a_{i_1} \otimes \cdots \otimes a_{i_m} = \bigotimes_{i=1}^{n} a_i^{m_i}$.
- Multiple alternate ways to get new patterns (e.g., because two or more nodes have the same degree) are combined with $\cup$.

The corresponding summand for k is created according to the following rules:

- $x \cup y \mapsto x + y$
- $\bigotimes_{i=1}^{n} a_i^{m_i} \mapsto \prod_{i=1}^{n} \binom{l_i}{m_i}$

For our example in figure 1.3 we have three possible rootings which we can see in figure 1.5. These correspond to the classes t_3, t_5, and t_7, which correspond to the disjoint classes a_3, a_2, and a_1 (in general this will be more involved).

- a_1: Each t_6-subpattern attached to a node of degree 5 produces a new pattern. The corresponding a-term is $a_2 \cup a_6$, and this results in the addition of the term $l_2 + l_6$ to $k(5, l_0, \ldots, l_7)$.

- a_2: Each pair of t_1 and t_4 attached to a node of degree 3 produces a new pattern. (Please note that in the case of planted roots, there is only space for one pair, since the third edge is needed to connect the planted root.) The a-term in this case is $(a_1 \cup a_7) \otimes (a_3 \cup a_4)$, and $l_1 l_3 + l_1 l_4 + l_3 l_7 + l_4 l_7$ gets added to $k(3, l_0, \ldots, l_7)$.

- a_3: Each t_2 attached to a node of degree 4 produces a new pattern. The a-term is $a_2 \cup a_5$, and we thus add $l_2 + l_5$ to $k(4, l_0, \ldots, l_7)$.

Thus $k(d, l_0, \ldots, l_7)$ is only non-zero for d equal to 3, 4, or 5, with values as specified above.

The next step is to convert the a_i descriptions, which we got as the result of the uniquification algorithm 1.3.2, to polynomials.

Convert to a polynomial using the following rule:

$$\bigotimes_{i=0}^{L-1} a_i^{m_i} \mapsto x u^{k(1+\sum m_i, m_0, \ldots, m_{L-1})} \prod_{i=0}^{L-1} \frac{a_i^{m_i}}{m_i!} \tag{1.2}$$

Note: Please observe in particular that 1 gets added to the node's degree (in the first argument of k) for the edge planting the root, which can not be used to create patterns but counts towards the total degree. Compare also with equation (1.5) below.

Now set

$$P_j(y_0, \ldots, y_{L-1}, u) = \sum_{l_0, \ldots, l_{L-1}} I_j(j, l_0, \ldots, l_{L-1}) u^{k(1+\sum_{i=0}^{L-1} l_i, l_0, \ldots, l_{L-1})} \frac{y_0^{l_0} \cdots y_{L-1}^{l_{L-1}}}{l_0! \cdots l_{L-1}!},$$

for $1 \le j \le L-1$. All subpatterns that are part of a match are handled in the P_j, $1 \le j \le L-1$, since P_0 only covers the tree types not occurring as subtrees in the pattern, and thus

$$P_0(y_0, \ldots, y_{L-1}, u) = e^{y_0 + \cdots + y_{L-1}} - \sum_{j=1}^{L-1} P_j(y_0, \ldots, y_{L-1}, 1)$$

does not depend on u.

Finally, let $a_{j;n,m}$ denote the number of planted rooted trees of type a_j with n nodes and m occurrences of the pattern $\mathcal{M}$ and set

$$a_j(x, u) = \sum_{n,m=0}^{\infty} a_{j;n,m} \frac{x^n u^m}{n!}.$$

By this definition it is clear that

$$a_j(x,u) = x \cdot P_j(a_0(x,u), \ldots, a_{L-1}(x,u), u).$$

Hence, we obtain the proposed structure of the system of functional equations (1.1).

In our example, after converting the descriptions from above into functions, we arrive at the following system of equations for the $a_i(x,u)$:

$$\begin{aligned}
a_0(x,u) = a_0 &= x + x\sum_{i=0}^{7} a_i + \frac{1}{2}x(a_0+a_2+a_5+a_6)^2 + x\sum_{n=5}^{\infty}\frac{1}{n!}\left(\sum_{i=0}^{7} a_i\right)^n \\
a_1(x,u) = a_1 &= x\left(\frac{1}{24}u^4(a_2+a_6)^4 + \right. \\
&\quad + \frac{1}{6}u^3(a_2+a_6)^3\,(a_0+a_1+a_3+a_4+a_5+a_7) + \\
&\quad + \frac{1}{4}u^2(a_2+a_6)^2\,(a_0+a_1+a_3+a_4+a_5+a_7)^2 + \\
&\quad \left. + \frac{1}{6}u(a_2+a_6)\,(a_0+a_1+a_3+a_4+a_5+a_7)^3\right) \\
a_2(x,u) = a_2 &= ux(a_3+a_4)(a_1+a_7) \\
a_3(x,u) = a_3 &= x\left(\frac{1}{6}u^3(a_2+a_5)^3 + \right. \\
&\quad + \frac{1}{2}u^2(a_2+a_5)^2(a_0+a_1+a_3+a_4+a_6+a_7) + \\
&\quad \left. + \frac{1}{2}u(a_2+a_5)(a_0+a_1+a_3+a_4+a_6+a_7)^2\right) \\
a_4(x,u) = a_4 &= \frac{1}{6}x(a_0+a_1+a_3+a_4+a_6+a_7)^3 \\
a_5(x,u) = a_5 &= x\left(\frac{1}{2}(a_1+a_7)^2 + (a_1+a_7)(a_0+a_2+a_5+a_6)\right) \\
a_6(x,u) = a_6 &= x\left(\frac{1}{2}(a_3+a_4)^2 + (a_3+a_4)(a_0+a_2+a_5+a_6)\right) \\
a_7(x,u) = a_7 &= \frac{1}{24}x\,(a_0+a_1+a_3+a_4+a_5+a_7)^4\,.
\end{aligned}$$

In order to complete the proof of proposition 1 we just have to show that the dependency graph is strongly connected. By construction, a_0 depends on all functions a_i. Thus, it is sufficient to prove that every a_i $(1 \le i < L)$ also depends on a_0. For this purpose consider the subtree of $\mathcal{M}$ that was labeled by a_i and consider a path from the root to an empty node. Each edge of this path corresponds to another subtree of $\mathcal{M}$, say $a_{i_2}, a_{i_3}, \ldots, a_{i_r}$. Then, by construction of the system of functional equations above, a_i depends on a_{i_2}, a_{i_2} depends on

a_{i_3} etc. Finally the root of a_{i_r} is adjacent to an empty node and thus (the corresponding generating function) depends on a_0. This completes the proof of proposition 1.

The next step is to find equations for the exponential generating function of rooted trees (where occurrences of the pattern are marked with u). We set

$$r(x,u) = \sum_{n,m=0}^{\infty} r_{n,m} \frac{x^n u^m}{n!},$$

where $r_{n,m}$ denotes the number of rooted trees of size n with exactly m occurrences of the pattern $\mathcal{M}$.

Proposition 2. *(***Rooted Trees***) Let $\mathcal{M}$ be a pattern and let*

$$a_0(x,u), \ldots, a_{L-1}(x,u)$$

denote the auxiliary functions introduced in proposition 1. Then there exists a polynomial $Q(y_0, \ldots, y_{L-1}, u)$ with non-negative coefficients and $Q(y_0, \ldots, y_{L-1}, 1) \leq_c e^{y_0+\cdots+y_{L-1}}$ such that

$$r(x,u) = G(x, u, a_0(x,u), \ldots, a_{L-1}(x,u)), \tag{1.3}$$

where

$$G(x, u, y_0, \ldots, y_{L-1}) = xQ(y_0, \ldots, y_{L-1}, u) + x\left(e^{\sum_{i=0}^{L-1} y_i} - Q(y_0, \ldots, y_{L-1}, 1)\right). \tag{1.4}$$

In principle, the proof directly continues the proof of proposition 1. We recall that a rooted tree is just a root with zero, one, two, ... planted subtrees, i.e., it can be described as a disjoint union of rooted trees of the form $a_{j_1} \otimes \cdots \otimes a_{j_d}$. Further, set

$$Q_d(y_0, \ldots, y_{L-1}, u) = \sum_{l_0,\ldots,l_{L-1}} I_d(l_0, \ldots, l_{L-1}) u^{k(d,l_0,\ldots,l_{L-1})} \frac{y_0^{l_0} \cdots y_{L-1}^{l_{L-1}}}{l_0! \cdots l_{L-1}!}. \tag{1.5}$$

Then by construction

$$r(x,u) = x \sum_{d \geq 0} Q_d(a_0(x,u), \ldots, a_{L-1}(x,u), u).$$

Note that $\sum_{d\geq 0} Q_d(y_0, \ldots, y_{L-1}, 1) = e^{y_0+\cdots+y_{L-1}}$. Let D denote the set of degrees of the internal (filled) nodes of the pattern; then $Q_d(y_0, \ldots, y_{L-1}, u)$ does not depend on u if d is different from all degrees in D. With

$$Q(y_0, \ldots, y_{L-1}, u) := \sum_{d \in D} Q_d(y_0, \ldots, y_{L-1}, u)$$

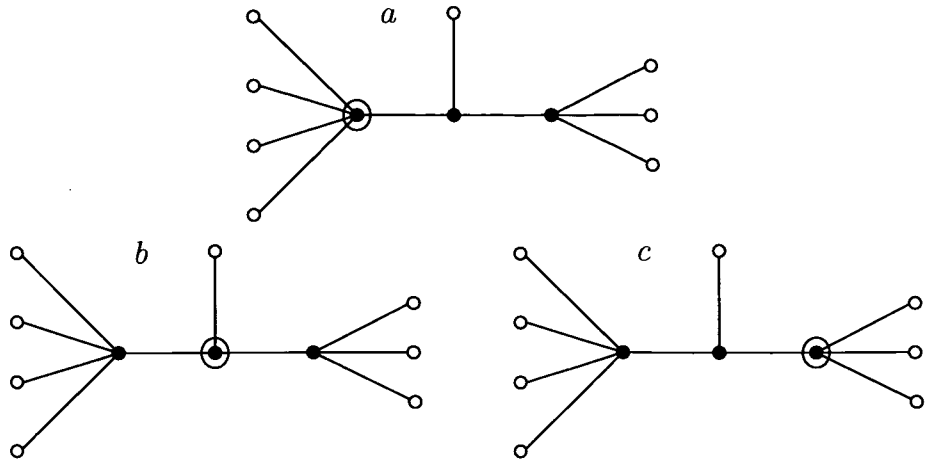

Figure 1.6: Possible roots in the pattern

we obtain (1.3) and (1.4).

Again, please note the difference of the first argument of $k(d, l_0, \ldots, l_{L-1})$ of equation (1.5) to equation (1.2) above. There, 1 was added to the first argument for the edge to the planted root, which is not needed here.

We illustrate the proof with our example. For convenience, let $r_0 = r_0(x, u)$ denote the function

$$r_0 = xe^p - \frac{xp^5}{5!} - \frac{xp^4}{4!} - \frac{xp^3}{3!},$$

where $p = a_0 + \cdots + a_7$. r_0 might be also interpreted as a catch-all function for the "uninteresting" subtrees – just a root x with an unspecified number of planted trees attached, except the ones we handle differently, namely the cases $d \in D = \{3, 4, 5\}$. The generating function $r = r(x, u)$ for rooted trees is then given by

$$\begin{aligned} r = r(x,u) = & r_0 + x \sum_{\substack{\sum m_i = 5 \\ m_i \geq 0}} u^{m_2+m_4} \frac{\prod a_i^{m_i}}{\prod m_i!} + \\ & + x \sum_{\substack{\sum m_i = 3 \\ m_i \geq 0}} u^{(m_1+m_5)(m_3+m_6)} \frac{\prod a_i^{m_i}}{\prod m_i!} + x \sum_{\substack{\sum m_i = 4 \\ m_i \geq 0}} u^{m_4+m_7} \frac{\prod a_i^{m_i}}{\prod m_i!}, \end{aligned}$$

compare also with figure 1.6.

As above we have $t_{n,m} = r_{n,m}/n$, where $t_{n,m}$ denotes the number of **unrooted trees** with n nodes and exactly m occurrences of the pattern $\mathcal{M}$. This relation is sufficient for our purposes. It is easy to express the corresponding generating

function $t(x,u)$ by

$$\begin{aligned} t(x,u) &= r(x,u) - \frac{1}{2}p(x,u)^2 \\ &\quad - \frac{u-1}{2}\big((a_1(x,u)+a_7(x,u))(a_2(x,u)+a_6(x,u) \\ &\quad + (a_2(x,u)+a_5(x,u))(a_3(x,u)+a_4(x,u))\big). \end{aligned}$$

The *correction term* comes from pairs of planted rooted trees that produce an additional pattern $\mathcal{M}$ when the planted roots are grafted together – (t_1, t_6) and (t_2, t_4) are only two pairs of trees where an additional pattern appears.

1.4 Asymptotic Behavior

Since we are not interested in the actual number of occurrences of the pattern, but only in the asymptotic behavior, we do not have to compute explicit formulae from the system of equations, but instead will apply a slightly adapted result from [Drm97], see section 1.4.1.

The theorem we want to prove in this section is:

Theorem 1. *Let $\mathcal{M}$ be a given finite tree. Then the limiting distribution of the number of occurrences of $\mathcal{M}$ (as induced subtrees) in a tree of $\mathcal{T}_n$ is asymptotically normal with mean $\sim \mu n$ and variance $\sim \sigma^2 n$, where $\mu > 0$ and $\sigma \geq 0$ depend on the pattern $\mathcal{M}$.*

1.4.1 Asymptotics of Analytic Systems

The following theorem is a slightly modified version of the main theorem from [Drm97]. Let $\mathbf{F}(x,\mathbf{y},\mathbf{u}) = (F_1(x,\mathbf{y},\mathbf{u}),\dots,F_N(x,\mathbf{y},\mathbf{u}))'$ be a vector of functions $F_j(x,\mathbf{y},\mathbf{u})$, $1 \leq j \leq N$, with complex variables x, $\mathbf{u} = (z_1,\dots,z_k)'$, $\mathbf{y} = (y_1,\dots,y_N)'$ which are analytic around 0 and satisfy $F_j(0,\mathbf{0},\mathbf{0}) = 0$, $1 \leq j \leq N$. We will be interested in the analytic solution $\mathbf{y} = \mathbf{y}(x,\mathbf{u}) = (y_1(x,\mathbf{u}),\dots,y_N(x,\mathbf{u}))'$ of the functional equation

$$\mathbf{y} = \mathbf{F}(x,\mathbf{y},\mathbf{u}) \tag{1.6}$$

with $\mathbf{y}(0,\mathbf{0}) = \mathbf{0}$, i.e., the (unknown) functions $y_j = y_j(x,\mathbf{u})$, $1 \leq j \leq N$, satisfy the system of functional equations

$$\begin{aligned} y_1 &= F_1(x,y_1,y_2,\dots,y_N,\mathbf{u}), \\ y_2 &= F_2(x,y_1,y_2,\dots,y_N,\mathbf{u}), \\ &\vdots \\ y_N &= F_N(x,y_1,y_2,\dots,y_N,\mathbf{u}). \end{aligned}$$

If the functions $F_j(x, \mathbf{y}, \mathbf{u})$ have non-negative Taylor coefficients then it is easy to see that the solutions $y_j(x, \mathbf{u})$ have the same property. (You only have to solve the system iteratively by setting $\mathbf{y}_0(x, \mathbf{u}) \equiv 0$ and $\mathbf{y}_{i+1}(x, \mathbf{u}) = \mathbf{F}(x, \mathbf{y}_i(x, \mathbf{u}), \mathbf{u})$ for $i \geq 0$. The limit $\mathbf{y}(x, \mathbf{u}) = \lim_{i \to \infty} \mathbf{y}_i(x, \mathbf{u})$ is the (unique) solution of the system above.)

Now suppose that $G(x, \mathbf{y}, \mathbf{u})$ is another analytic function with non-negative Taylor coefficients. Then $G(x, \mathbf{y}(x, \mathbf{u}), \mathbf{u})$ has a power series expansion

$$G(x, \mathbf{y}(x, \mathbf{u}), \mathbf{u}) = \sum_{n, \mathbf{m}} c_{n,\mathbf{m}} x^n \mathbf{u}^{\mathbf{m}}$$

with non-negative coefficients $c_{n,\mathbf{m}}$. In fact, we will assume that for every $n \geq n_0$ there exists $\mathbf{m}$ such that $c_{n,\mathbf{m}} > 0$.

Let $\mathbf{X}_n$ $(n \geq n_0)$ denote an N-dimensional discrete random vector with

$$\mathbf{Pr}[\mathbf{X}_n = \mathbf{m}] := \frac{c_{n,\mathbf{m}}}{c_n}, \tag{1.7}$$

where

$$c_n = \sum_{\mathbf{m}} c_{n,\mathbf{m}}$$

are the coefficients of

$$G(x, \mathbf{y}(x, \mathbf{1}), \mathbf{1}) = \sum_{n \geq 0} c_n x^n.$$

The following theorem shows that (under suitable analyticity conditions) $\mathbf{X}_n$ has a Gaussian limiting distribution.

Theorem 2. *Let* $\mathbf{F}(x, \mathbf{y}, \mathbf{u}) = (F_1(x, \mathbf{y}, \mathbf{u}), \ldots, F_N(x, \mathbf{y}, \mathbf{u}))'$ *be analytic functions around* $x = 0$, $\mathbf{u} = (u_1, \ldots, u_N)' = \mathbf{0}$, $\mathbf{y} = (y_1, \ldots, y_N)' = \mathbf{0}$ *such that all Taylor coefficients are non-negative, that* $\mathbf{F}(0, \mathbf{u}, \mathbf{y}) \equiv \mathbf{0}$, *that* $\mathbf{F}(x, \mathbf{u}, \mathbf{0}) \not\equiv \mathbf{0}$, *and that there exists* j *with* $\mathbf{F}_{y_j y_j}(x, \mathbf{y}, \mathbf{u}) \not\equiv \mathbf{0}$. *Furthermore assume that the region of convergence of* $\mathbf{F}$ *is large enough such that there exists a non-negative solution* $x = x_0, \mathbf{y} = \mathbf{y}_0$ *of the system of equations*

$$\begin{aligned} \mathbf{y} &= \mathbf{F}(x, \mathbf{y}, \mathbf{1}), \\ 0 &= \det(\mathbf{I} - \mathbf{F}_{\mathbf{y}}(x, \mathbf{y}, \mathbf{1})), \end{aligned}$$

inside it. Let

$$\mathbf{y} = \mathbf{y}(x, \mathbf{u}) = (y_1(x, \mathbf{u}), \ldots, y_N(x, \mathbf{u}))'$$

denote the analytic solutions of the system

$$\mathbf{y} = \mathbf{F}(x, \mathbf{y}, \mathbf{u}) \tag{1.8}$$

and assume that $c_{n,j} > 0$ $(1 \leq j \leq N)$ *for* $n \geq n_0$, *where* $y_j(x, \mathbf{1}) = \sum_{n \geq 0} c_{n,j} x^n$. *Moreover, let* $G(x, \mathbf{y}, \mathbf{u})$ *denote an analytic function with non-negative Taylor*

coefficients such that the point $(x_0, \mathbf{y}(x_0, \mathbf{1}), \mathbf{1})$ *is contained in the region of convergence. Finally, let the random vector* $\mathbf{X}_n$ $(n \geq n_0)$ *be defined by (1.7).*

If the dependency graph $G_{\mathbf{F}} = (V, E)$ *of the system (1.8) in the unknown functions* $y_1(x, \mathbf{u}), \ldots, y_N(x, \mathbf{u})$ *is strongly connected then the sequence of random vectors* $\mathbf{X}_n$ *admits a Gaussian limiting distribution with mean value*

$$\mathbf{E}\,\mathbf{X}_n = \mu\, n + O(1) \qquad (n \to \infty)$$

and covariance matrix

$$\mathbf{Cov}(\mathbf{X}_n, \mathbf{X}_n) = \Sigma\, n + O(1) \qquad (n \to \infty).$$

The vector μ *is given by*

$$\mu = -\frac{x_{\mathbf{u}}(\mathbf{1})}{x(\mathbf{1})},$$

and the matrix Σ *by*

$$\Sigma = -\frac{x_{\mathbf{uu}}(\mathbf{1})}{x(\mathbf{1})} + \mu'\mu + \operatorname{diag}(\mu),$$

where $x = x(\mathbf{u})$ *(and* $\mathbf{y} = \mathbf{y}(\mathbf{u}) = \mathbf{y}(x(\mathbf{u}), \mathbf{u})$*) is the solution of the (extended) system*

$$\mathbf{y} = \mathbf{F}(x, \mathbf{y}, \mathbf{u}), \tag{1.9}$$

$$0 = \det(\mathbf{I} - \mathbf{F}_{\mathbf{y}}(x, \mathbf{y}, \mathbf{u})). \tag{1.10}$$

The proof of theorem 2 is exactly the same as that given in [Drm97]. The main observation is that the assumptions above show that the solutions $y_j(x, \mathbf{u})$ admit a local representation of the form

$$y_j(x, \mathbf{u}) = g_j(x, \mathbf{u}) - h_j(x, \mathbf{u})\sqrt{1 - \frac{x}{x(\mathbf{u})}},$$

(where $\mathbf{u}$ is close to $\mathbf{1}$ and x close to $x_0 = x(\mathbf{1})$). The assumption that the dependency graph is strongly connected ensures that the location of the singularity of all functions $y_j(x, \mathbf{u})$ is determined by the common function $x(\mathbf{u})$. Thus, we get the same property for $G(x, \mathbf{y}(x, \mathbf{u}), \mathbf{u})$:

$$G(x, \mathbf{y}(x, \mathbf{u}), \mathbf{u}) = g(x, \mathbf{u}) - h(x, \mathbf{u})\sqrt{1 - \frac{x}{x(\mathbf{u})}} \tag{1.11}$$

It is then well known (see [BR83a], [Drm94a]) that a square-root singularity (plus some minor conditions; note, for example, that the assumption $c_{n,j} > 0$ for $n \geq n_0$ ensures that $x_0 = x(\mathbf{1})$ is the only singularity on the radius of convergence of $G(x, \mathbf{y}(x, \mathbf{1}), \mathbf{1})$ and that $c_n > 0$ for sufficiently large n implies asymptotic normality of the coefficients (in the sense introduced above) with mean and covariance expressed in terms of derivatives of $x(\mathbf{u})$.

In what follows we comment on the evaluation of μ and Σ. The problem is to extract the derivatives of $x(\mathbf{u})$, which is the solution of the system (1.9), (1.10) and is exactly the location of the singularity of the mapping $x \mapsto \mathbf{y}(x, \mathbf{u})$ when $\mathbf{u}$ is fixed (and is close to $\mathbf{1}$).

Let $x(\mathbf{u})$ and $\mathbf{y}(\mathbf{u}) = \mathbf{y}(x(\mathbf{u}), \mathbf{u})$ denote the solutions of (1.9), (1.10). Then we have

$$\mathbf{y}(\mathbf{u}) = \mathbf{F}(x(\mathbf{u}), y(\mathbf{u}), \mathbf{u}). \tag{1.12}$$

Taking derivatives with respect to $\mathbf{u}$ we get

$$\mathbf{y_u}(\mathbf{u}) = \mathbf{F}_x(x(\mathbf{u}), y(\mathbf{u}), \mathbf{u})x_\mathbf{u}(\mathbf{u}) + \mathbf{F_y}(x(\mathbf{u}), y(\mathbf{u}), \mathbf{u})\mathbf{y_u}(\mathbf{u}) + \mathbf{F_u}(x(\mathbf{u}), y(\mathbf{u}), \mathbf{u})x_\mathbf{u}(\mathbf{u}). \tag{1.13}$$

In particular, for $\mathbf{u} = \mathbf{1}$ we have $x(\mathbf{1}) = x_0$ and $\mathbf{y}(\mathbf{1}) = \mathbf{y}_0$ and, of course

$$\det(\mathbf{I} - \mathbf{F_y}(x_0, \mathbf{y}_0, \mathbf{1})) = 0.$$

Since $\mathbf{F_y}$ is a non-negative matrix and the dependency graph is strongly connected there is a unique positive eigenvalue of multiplicity 1 which equals 1. Thus, $\mathbf{I} - \mathbf{F_y}$ has rank $N - 1$ and has (up to scaling) a unique (left) eigenvector $\mathbf{b}'$:

$$\mathbf{b}'(\mathbf{I} - \mathbf{F_y}(x_0, \mathbf{y}_0, \mathbf{1})) = \mathbf{0}.$$

From (1.13) we obtain

$$(\mathbf{I} - \mathbf{F_y}(x_0, \mathbf{y}_0, \mathbf{1}))\mathbf{y_u}(\mathbf{1}) = \mathbf{F}_x(x_0, \mathbf{y}_0, \mathbf{1})x_\mathbf{u}(\mathbf{1}) + \mathbf{F_u}(x_0, \mathbf{y}_0, \mathbf{1}).$$

By multiplying $\mathbf{b}'$ from the left we thus get

$$x_\mathbf{u}(\mathbf{1}) = -\frac{\mathbf{b}'\mathbf{F_u}(x_0, \mathbf{y}_0, \mathbf{1})}{\mathbf{b}'\mathbf{F}_x(x_0, \mathbf{y}_0, \mathbf{1})}$$

and consequently

$$\mu = \frac{1}{x_0}\frac{\mathbf{b}'\mathbf{F_u}(x_0, \mathbf{y}_0, \mathbf{1})}{\mathbf{b}'\mathbf{F}_x(x_0, \mathbf{y}_0, \mathbf{1})} \tag{1.14}$$

The derivation of Σ is much more involved. We have to derivate (1.12) twice to get some information for $x_{\mathbf{uu}}(\mathbf{1})$. However, we also have to know $\mathbf{y_u}(\mathbf{1})$, which can be calculated from the first derivative of (1.12) and from the derivative of

$$\det(\mathbf{I} - \mathbf{F_y}(x(\mathbf{u}), \mathbf{y}(\mathbf{u}), \mathbf{1}) = 0.$$

This is very messy. Even if the *system* of equations consists just of one equation and $\mathbf{u} = u$ is one-dimensional we get a rather involved formula, compare with [Drm94a].

1.4.2 Applying the Theorem

It is immediately clear that theorem 2 applies to the kind of problem we are interested in. The assertions of propositions 1 and 2 by construction exactly fit the assumptions of theorem 2.

The only missing point is the existence of $x_0, \mathbf{a}_0$ of the system

$$\mathbf{a} = \mathbf{F}(x, \mathbf{a}, 1), \tag{1.15}$$

$$0 = \det(\mathbf{I} - \mathbf{F}_{\mathbf{a}}(x, \mathbf{a}, 1)). \tag{1.16}$$

Since the sum of all unknown functions $p(x, u)$ is known for $u = 1$:

$$p(x, 1) = p(x) = \sum_{n \geq 1} n^{n-1} \frac{x^n}{n!} = 1 - \sqrt{2}\sqrt{1 - ex} + \cdots,$$

it is not unexpected that $x_0 = 1/e$.

Proposition 3. *Let $\mathbf{a} = \mathbf{F}(x, \mathbf{a}, u)$ be the system of functional equations of proposition 1. Then $x_0, \mathbf{a}_0$ uniquely exist and $x_0 = 1/e$.*

Proof. Set $u = 1$ and consider the solution $\mathbf{a}(x, 1) = (a_0(x, 1), \ldots, a_{L-1}(x, 1))$. Since the dependency graph is strongly connected it follows that all functions $a_j(x, 1)$ have the same radius of convergence which has to be $x_0 = 1/e$, and all functions are singular at $x = x_0$. Since $a_j(x, 1) \leq p(x, 1)$ for $0 \leq x < x_0$ it also follows that $a_j(x_0, 1)$ is finite, and we have $\mathbf{a}(x_0, 1) = \mathbf{F}(x_0, \mathbf{a}(x_0, 1), 1)$. If $\det(\mathbf{I} - \mathbf{F}_{\mathbf{a}}(x_0, \mathbf{a}(x_0, 1), 1)) \neq 0$ then the implicit function theorem would imply that there is an analytic continuation for $a_j(x, 1)$ around $x = x_0$ which is, of course, a contradiction. Thus, the system above has a (unique) solution. □

Consequently, it follows that the numbers $r_{n,m}$ have a Gaussian limiting distribution with mean and variance which are proportional to n. Since $t_{n,m} = r_{n,m}/n$ we get exactly the same law for unrooted trees. This proves theorem 1.

Before we demonstrate this kind of technique with our example we state a formula for the mean value μ.

Proposition 4. *Let $x_0 = 1/e$ and $\mathbf{a}_0$ be given by proposition 3 and let $P_j(\mathbf{y}, u)$ $(1 \leq j \leq L - 1)$ be the polynomials of proposition 1. Then μ (of theorem 1) is given by*

$$\mu = \frac{1}{e} \sum_{j=1}^{L-1} \frac{\partial P_j}{\partial u}(\mathbf{a}_0, 1). \tag{1.17}$$

Proof. Formula (1.17) follows from (1.14). Since

$$\mathbf{F}(x,\mathbf{a},u)=\begin{pmatrix} x\left(e^{a_0+\cdots+a_{L-1}}-\sum_{j=1}^{L-1}P_j(\mathbf{a},u)\right)\\ xP_1(\mathbf{a},u)\\ xP_2(\mathbf{a},u)\\ \vdots\\ xP_{L-1}(\mathbf{a},u)\end{pmatrix}$$

we get

$$\mathbf{F}_{\mathbf{a}}=x\begin{pmatrix} e^{a_0+\cdots+a_{L-1}}-\sum_{j=1}^{L-1}P_{j,a_0} & \cdots & e^{a_0+\cdots+a_{L-1}}-\sum_{j=1}^{L-1}P_{j,a_{L-1}}\\ P_{1,a_0} & \cdots & P_{1,a_{L-1}}\\ \vdots & & \vdots\\ P_{L-1,a_0} & \cdots & P_{L-1,a_{L-1}}\end{pmatrix}.$$

Since $a_0(x_0,1)+\cdots+a_{L-1}(x_0,1)=p(x_0,1)=1$ we have $x_0e^{a_0(x_0,1)+\cdots a_{L-1}(x_0,1)}=1$. Consequently the sum of all rows of $\mathbf{F}_{\mathbf{a}}$ equals $(1,1,\ldots,1)$ for $x=x_0=1/e$. Thus, the vector $\mathbf{b}'=(1,1,\ldots,1)$ is the (up to scaling) unique left eigenvector of $\mathbf{I}-\mathbf{F}_{\mathbf{a}}$.

It is now easy to check that

$$x_0\mathbf{b}'\mathbf{F}_x(x_0,\mathbf{a}_0,1)=\frac{1}{e}e^{a_0(x_0,1)+\cdots a_{L-1}(x_0,1)}=1$$

and that

$$x_0\mathbf{b}'\mathbf{F}_u(x_0,\mathbf{a}_0,1)=\frac{1}{e}\sum_{j=0}^{L-1}P_{j,u}(\mathbf{a}_0,1).$$

This completes the proof of proposition 4. □

We now come back to our example and explicitly calculate the mean value μ. In a first step we have to evaluate our generating functions at $x_0=1/e$ and $u=1$, which involves solving a system of equations.

$$\begin{aligned} p(1/e,1)&=1,\\ a_0(1/e,1)&=1-\frac{5}{24e}-\frac{5}{24e^2}+\frac{25}{1152e^3},\\ a_1(1/e,1)&=\frac{1}{e}\left(\frac{1}{4!}\left(\frac{12e-1}{72e^3}\right)^4+\frac{1}{3!}\left(\frac{12e-1}{72e^3}\right)^3\left(1-\frac{12e-1}{72e^3}\right)+\right.\\ &\quad+\frac{1}{2!2!}\left(\frac{12e-1}{72e^3}\right)^2\left(1-\frac{12e-1}{72e^3}\right)^2+\\ &\quad\left.+\frac{1}{3!}\left(\frac{12e-1}{72e^3}\right)\left(1-\frac{12e-1}{72e^3}\right)^3\right), \end{aligned}$$

$$a_2(1/e,1) = \frac{1}{144e^3},$$
$$a_3(1/e,1) = \frac{1}{e}\left(\frac{1}{3!}\left(\frac{48e-1}{1152e^3}\right)^3 + \frac{1}{2!}\left(\frac{48e-1}{1152e^3}\right)^2\left(1-\frac{48e-1}{1152e^3}\right) + \right.$$
$$\left. + \frac{1}{2!}\left(\frac{48e-1}{1152e^3}\right)\left(1-\frac{48e-1}{1152e^3}\right)^2\right),$$
$$a_4(1/e,1) = \frac{1}{3!e}\left(1-\left(\frac{48e-1}{1152e^3}\right)\right)^3,$$
$$a_5(1/e,1) = \frac{16e-3}{384e^3},$$
$$a_6(1/e,1) = \frac{8e-1}{48e^3},$$
$$a_7(1/e,1) = \frac{1}{4!e}\left(1-\frac{12e-1}{72e^3}\right)^4.$$

Furthermore, we have

$$P_{0,u}(\mathbf{a}_0,1) = 0,$$
$$P_{1,u}(\mathbf{a}_0,1) = \frac{1}{3!}(a_2+a_6)^4 + \frac{1}{2!}(a_2+a_6)^3\left(a_0+a_1+a_4+\sum_{i=5}^{7}a_i\right) +$$
$$+ \frac{1}{2!}(a_2+a_6)^2\left(a_0+a_1+a_4+\sum_{i=5}^{7}a_i\right)^2 +$$
$$+ \frac{1}{3!}(a_2+a_6)\left(a_0+a_1+a_4+\sum_{i=5}^{7}a_i\right)^3$$
$$= \frac{12e-1}{432e^3},$$
$$P_{2,u}(\mathbf{a}_0,1) = 0,$$
$$P_{3,u}(\mathbf{a}_0,1) = 0,$$
$$P_{4,u}(\mathbf{a}_0,1) = (a_3+a_4)(a_1+a_7) = \frac{1}{144e^2},$$
$$P_{5,u}(\mathbf{a}_0,1) = 0,$$
$$P_{6,u}(\mathbf{a}_0,1) = \frac{1}{2!}(a_2+a_5)^3 + (a_2+a_5)^2(a_0+a_1+a_3+a_4+a_6+a_7) +$$
$$+ \frac{1}{2!}(a_2+a_5)(a_0+a_1+a_3+a_4+a_6+a_7)^2 = \frac{48e-1}{2304e^3},$$
$$P_{7,u}(\mathbf{a}_0,1) = 0.$$

Thus we get:

$$\mu = \frac{1}{e}\sum_{j=0}^{7} P_{j,u}(\mathbf{a}_0, 1) = \frac{384e - 19}{6912e^4} = 0.002715601434...$$

It is interesting to observe that (in this example) μ can be written as a rational polynomial in $1/e$. It is not obvious whether such a property remains true in general. Nevertheless, we can say that μ is algebraic in $1/e$ since the system (1.15) and (1.16) is an algebraic one.

1.5 Extensions and Generalizations

In what follows we list some obvious and some less obvious extensions.

1.5.1 Several Patterns

Let $\mathcal{M}_1, \ldots, \mathcal{M}_k$ be k different patterns. Then the problem is to determine the joint (limiting) distribution of the number of occurrences of $\mathcal{M}_1, \ldots, \mathcal{M}_k$ in trees of size n. Using the same techniques as above (splitting the patterns into planted subpatterns; see the algorithm in section 1.3.2) we again obtain a system of functional equations. The only difference is that we now have to count occurrences of $\mathcal{M}_1, \ldots, \mathcal{M}_k$ with different variables $u_1, \ldots, u_k$, which can however be done in the same fashion as for a single u. In view of theorem 2 multiple us make no difference and we obtain a multivariate Gaussian limiting distribution.

1.5.2 Filled and Empty Nodes

In our model we have distinguished between internal (filled) and external (empty) nodes of the pattern $\mathcal{M}$, where the degrees of the internal (filled) nodes have to match exactly. It is also possible to consider the following more general matching problem: Let $\mathcal{M}$ again be a finite tree where this time arbitrarily chosen nodes are *filled* and the remaining ones are *empty*. Now we say that $\mathcal{M}$ matches if it occurs as a subtree such that the corresponding degrees of the filled nodes are equal whereas the degrees of the empty nodes might be different. The counting procedure above can be adapted to cover this case, too, however, it is a little bit more involved. For example, if leaves of the pattern are filled nodes then these nodes have to be leaves wherever the pattern occurs. These kinds of situations also lead to systems of functional equations for which the dependency graph will not be strictly connected. More precisely, some of the functions $a_j(x, u)$ are explicitly given in the system. Nevertheless, by substituting these explicit expressions one gets a smaller system with a strongly connected dependency graph.

1.5.3 Pattern Containing Paths of Unspecified Length

It might also be interesting to consider patterns where specific edges can be replaced by paths of arbitrary length. It turns out that this case in particular is more involved since a *natural* partition of all planted rooted trees is now infinite – we are confronted with an infinite system of equations for the corresponding generating functions. In such a case theorem 2 cannot be applied any more.

Nevertheless, it seems that the approach of Lalley [Lal02] that is applicable for infinite systems of functional equations in one variable can be generalized to a corresponding generalization of theorem 2 to proper infinite systems. Thus, we can expect a Gaussian limit law even in this case.

1.5.4 Simply Generated Trees

Simply generated trees have been introduced by Meir and Moon [MM78] and are proper generalizations of several types of rooted trees. Let

$$\varphi(x) = \varphi_0 + \varphi_1 x + \varphi_2 x^2 + \cdots$$

be a power series with non-negative coefficients; in particular we assume that $\varphi_0 > 0$ and $\varphi_j > 0$ for some $j \geq 2$. We then define the weight $\omega(T)$ of a finite rooted tree T by

$$\omega(T) = \prod_{j \geq 0} \varphi_j^{D_j(T)},$$

where $D_j(T)$ denotes the number of nodes in T with j successors. If we set

$$y_n = \sum_{|T|=n} \omega(T)$$

then the generating function

$$y(x) = \sum_{n \geq 1} y_n x^n$$

satisfies the functional equation

$$y(x) = x\varphi(y(x)).$$

In this context y_n denotes a weighted number of trees is size n. For example, if $\varphi_j = 1$ for all $j \geq 0$ (that is, $\varphi(x) = 1/(1-x)$) then all rooted trees have weight $\omega(T) = 1$ and $y_n = p_n$ is the number of planted plane trees. If $\varphi_j = 1/j!$ (that is, $\varphi(x) = e^x$) then we formally get labeled rooted trees etc.

We can proceed in the same way as above and obtain a system of functional equations that counts occurrences of a specific pattern in simply generated trees, and (under suitable conditions on the growth of φ_j) we finally obtain a Gaussian limiting distribution.

1.5.5 Unlabeled Trees

Let $\hat{p}_n$ denote the number of unlabeled planted rooted trees and $\hat{t}_n$ the number of unlabeled unrooted trees. The generating functions are denoted by

$$\hat{p}(x) = \sum_{n\geq 1} \hat{p}_n x^n \quad \text{and} \quad \hat{t}(x) = \sum_{n\geq 1} \hat{t}_n x^n.$$

The structure of these trees is much more difficult than that of labeled trees. It turns out that one has to apply Pólya's theory of counting and an amazing observation (1.18) by Otter [Ott48]. The generating functions $\hat{p}(x)$ and $\hat{t}(x)$ satisfy the functional equations

$$\begin{aligned}\hat{p}(x) &= x \sum_{k\geq 0} Z(S_k; \hat{p}(x), \hat{p}(x^2), \ldots, \hat{p}(x^k)) \\ &= x \exp\left(\hat{p}(x) + \frac{1}{2}\hat{p}(x^2) + \frac{1}{3}\hat{p}(x^3) + \cdots\right)\end{aligned}$$

and

$$\hat{t}(x) = \hat{p}(x) - \frac{1}{2}\hat{p}(x)^2 + \frac{1}{2}\hat{p}(x^2), \tag{1.18}$$

where $Z(S_k; x_1, \ldots, x_k)$ denotes the cycle index of the symmetric group S_k. These functions have a common radius of convergence $\rho \approx 0.338219$ and local expansions of the form

$$\hat{p}(x) = 1 - b(\rho - x)^{1/2} + c(\rho - x) + d(\rho - x)^{3/2} + \mathcal{O}\left((\rho - x)^2\right)$$

and

$$\hat{t}(x) = \frac{1 + \hat{p}(\rho^2)}{2} + \frac{b^2 - \rho\hat{p}'(\rho^2)}{2}(\rho - x) + bc(\rho - x)^{3/2} + \mathcal{O}\left((\rho - x)^2\right),$$

where $b \approx 2.6811266$ and $c = b^2/3 \approx 2.3961466$, and $x = \rho$ is the only singularity on the circle of convergence $|x| = \rho$. Thus, they behave similarly as $p(x)$ and $t(x)$. We also get

$$\hat{p}_n = \frac{b\sqrt{\rho}}{2\sqrt{\pi}} n^{-3/2} \rho^{-n} \left(1 + \mathcal{O}\left(n^{-1}\right)\right)$$

and

$$\hat{t}_n = \frac{b^3 \rho^{3/2}}{4\sqrt{\pi}} n^{-5/2} \rho^{-n} \left(1 + \mathcal{O}\left(n^{-1}\right)\right).$$

Furthermore, it is possible to count the number of nodes of specific degree with help of bivariate generating functions (compare with [DG99]). Thus, with help of Pólya's theory of counting we can also obtain a system of functional equations for bivariate generating functions that count the number of occurrences of a specific pattern. The major difference to the procedure above is that this system will also contain terms of the form $a_j(x^k, u^k)$ for $k \geq 2$. Fortunately these terms can be considered as *known functions* when x varies around the singularity ρ and u varies around 1 (compare again with [DG99]). Hence, theorem 2 applies again and we can proceed as above.

1.5.6 Forests

First, let us consider the case of labeled trees with generating function $t(x,u)$. Then the generating function $f(x,u)$ of unlabeled forests is given by

$$f(x,u) = e^{t(x,u)}.$$

Thus, the singular behavior of $f(x,u)$ is the same as that of $t(x,u)$ (compare with [DG99]) and consequently we again obtain a Gaussian limiting distribution for the number of occurrences of a specific pattern in labeled forests.

The case of unlabeled forests is similar. Here we have

$$\hat{f}(x,u) = \exp\left(\hat{t}(x,u) + \frac{1}{2}\hat{t}(x^2,u^2) + \frac{1}{3}\hat{t}(x^3,u^3) + \cdots\right).$$

Of course, we can consider other classes of trees or forests of a given number of trees.

1.5.7 Forbidden Patterns

It is also interesting to count the number $t_{n,0}$ of trees of size n without a given pattern. The generating function of these numbers is just $p(x,0)$ resp. $t(x,0)$. One can show that there exists an $\eta > 0$ such that

$$t_{n,0} \leq t_n e^{-\eta n}.$$

The only thing we have to check is that the radius of convergence of $t(x,0)$ is larger than the radius of convergence of $t(x,1)$. However, this is obvious since the radius of convergence of $t(x,u)$ (which is the same as that of $p(x,u)$) is given by $x(u)$ (for u around 1) and $x'(1) < 0$.

1.6 MAPLE Source Code

We provide a MAPLE program of an algorithm that generates the corresponding system of equations for a given pattern tree. When this system is solved, μ can be computed easily.

The implementation provides a MAPLE module named *treepattern* with a single user entry points: *TreeToEquations*.

TreeToEquations(tree) returns the corresponding system of equations for the tree pattern given by *tree*, which is expected in the grammar described by *combstruct[specification]* from the *algolib combstruct* package (see `http://algo.inria.fr/encyclopedia/help.html#specification` or the MAPLE online help after installing *algolib*).

The MAPLE code included below is a modified and extended version of code originally by Frédéric Chyzak.

1.6.1 Usage

A sample session testing the example from figure 1.2 might run as follows:

```
# maple8
    |\^/|     Maple 8 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) 2002 by Waterloo Maple Inc.
 \  MAPLE  /  All rights reserved. Maple is a registered trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(treepattern);
                                [TreeToEquations]

> tree:=Prod(Z[1],Set(
>     Prod(Z[2],Set(Prod(Z[i],Epsilon)$i=3..5)),
>     Prod(Z[6],Epsilon),
>     Prod(Z[7],Set(Prod(Z[i],Epsilon)$i=8..11))
> ));
tree := Prod(Z[1], Set(Prod(Z[2], Set(Prod(Z[3], Epsilon),

    Prod(Z[4], Epsilon), Prod(Z[5], Epsilon))),

    Prod(Z[6], Epsilon), Prod(Z[7], Set(Prod(Z[8], Epsilon),

    Prod(Z[9], Epsilon), Prod(Z[10], Epsilon),

    Prod(Z[11], Epsilon)))))

> eqnew:=TreeToEquations(tree):
bytes used=4000192, alloc=3669344, time=0.44
>
```

The resulting system of equations is not included here by purpose, since it would fill 11 pages; it does, however, give the same system that we computed earlier on page 23.

Additional informational and debugging messages can be enabled by setting *infolevel[treepattern]* to a value from 1 to 5, where a higher number means more debugging output.

1.6.2 Implementation

TreeToEquations(tree) returns the corresponding system of equations for the tree given in the *tree* argument.

The function is split up in many smaller internal functions for better testing and understanding. *TreeToEquations* itself initializes some module-specific variables and then calls the following functions in order:

- *TreeToPatterns* takes a tree as input and labels it recursively by using the *RecursiveLabeling* function, which also fills in the *adjacencyTable*. The result is normalized, that is, each unique subtree gets a name $t[i]$. Additionally, it creates a list *newPatterns* which contains all possible ways the pattern can appear during the recursive creation of all trees.
- *PatternsToPartition* converts the *adjacencyTable* into a recursive description of $t[i]$s by other $t[j]$s.
- *PartitionToSystem* implements the algorithm described in section 1.3.2, that is, it creates a partition $a[i]$ of all trees that is a fine enough to describe all $t[i]$s. It calls *Summary* to report information about the work it does at *infolevel* 2.
- *NewPatternsToList* converts *newPatterns* into the form used by *SystemToEquations*.
- *SystemToEquations* creates the function k described in section 1.3.2 and uses it to create the system of equations.

The remaining helper functions and internal variables are commented in the MAPLE code itself:

```
# -*- maple -*-
#
# $Id: treepattern.maple,v 1.20 2004/05/14 22:24:48 wiz Exp $
#
#
# Functions are explained where they are defined.
#
# Variable explanations:
# adjacencyTable: table of list of neighbor nodes
# assocTable: table mapping t[i] to list of neighbor nodes
# aPartition: table containing descriptions of a and t in a
# newPatterns: list of normalized trees causing new pattern
#     emergence; form: [t[i_1], ..., t[i_n]]
# newPattersList: one-level expanded version of newPatterns;
#     form:  [[a_11, a_12, ...], ..., [a_n1, a_n2, ...]]
# patternNumber: number of t-patterns
# tPartition: table containing descriptions of t in t
# visitedTable: for recursive labeling; mark visited nodes
```

```
treepattern:=module()
    export
          TreeToEquations;
    local
          adjacencyTable,
          assocTable,
          aPartition,
          newPatterns,
          newPatternsList,
          patternNumber,
          tPartition,
          visitedTable,
          AddNewA,
          AddNewT,
          AddToAssocTable,
          ExpandTree,
          ExpandTreeOneLevel,
          FindInAssocTable,
          NewPatternsToList,
          NormalizeTree,
          OriginalTToTensorsOfA,
          PartitionSetListProduct,
          PartitionSetProduct,
          PartitionToSystem,
          PatternsToPartition,
          RecursiveLabeling,
          SplitSomeA,
          Summary,
          SystemToEquations,
          TraverseAndNormalize,
          TreeCompare,
          TreeToPatterns,
          NONE,
          NOT_IN_TABLE;
    option package;


    # Create a new a[n] with value v and return n.
    AddNewA:=proc(v)
        local n;

        n:=aPartition["a", "size"]+1;
```

```
    userinfo(2, treepattern, "created a["||n||"]");
    aPartition["a", n]:=v;
    aPartition["a", "size"]:=n
end proc:

# Create a new t[n] with value v and return n.
AddNewT:=proc(v)
    local n;

    n:=aPartition["t","size"]+1;
    userinfo(2, treepattern, "created t["||n||"]");
    aPartition["t",n]:=v;
    aPartition["t","size"]:=n
end proc:

# Look up tree in assocTable; return its name if found,
# or create a new name for it. tree has to be normalized.
AddToAssocTable:=proc(tree)
    local found, s;

    # look tree up in existing names and return if found
    found:=FindInAssocTable(tree);
    if found <> NOT_IN_TABLE then
        return found
    end if;

    # create new name
    s:=patternNumber;
    assocTable[s]:=tree;
    patternNumber:=s+1;

    # return new name
    t[s]
end proc;

# Replace t[i] with assocTable[i].
ExpandTreeOneLevel:=proc(tree)
    local i;

    i:=op(tree);
    if i >= patternNumber then
        userinfo(5, treepattern,
                 "i: "||i||", patternNumber: "
```

```
                    ||patternNumber);
            error("bug!")
        end if;
        assocTable[i]
    end proc;

    # Recursively expand tree with ExpandTreeOneLevel.
    ExpandTree:=proc(tree)
        option remember;

        map(procname, ExpandTreeOneLevel(tree))
    end proc;

    # Look up a tree in assocTable and return the corresponding
    # tree, or NOT_IN_TABLE.
    FindInAssocTable:=proc(tree)
        local i;

        for i from 0 to patternNumber do
            if assocTable[i] = tree then
                return t[i]
            end if
        end do;
        NOT_IN_TABLE
    end proc;

    # Convert newPatterns in the form {t[a_i],...t[a_1]} to
    # list of neighbours in the form [[a_i_1, a_i_2, ...], ...,
    # [a_1_1, a_1_2, ...]]
    NewPatternsToList:=proc()
        local i, j;

        newPatternsList:=[
            seq([seq(op([j,1], [op(assocTable[op([i,1],
                                    newPatterns)])]),
                    j=1..nops(assocTable[op([i,1],
                                        newPatterns)]))],
                i=1..nops(newPatterns))];

        userinfo(5, treepattern, newPatternsList);
    end proc;

    # Normalize tree by sorting its children by size, using
```

```
# TreeCompare; adds normalized tree to assocTable by calling
# AddToAssocTable. Takes children as input.
NormalizeTree:=proc()
    option remember;

    AddToAssocTable(Tree(op(sort([args],TreeCompare))))
end proc;

# Express t[n+1] in terms of a set of lists of a[1..n].
# Example:
# If tPartition[3] is [2, 0, 0] and aPartition["t", 2]
# is {2, 5}, then OriginalTToTensorsOfA gives
# {[2, 0, 0], [5, 0, 0]}.
OriginalTToTensorsOfA:=proc()
    local n,i;

    n:=aPartition["t","size"];
    PartitionSetListProduct(subs(aPartition["t", 0]
                                 ={seq(i, i=0..n)},
                            map(i->aPartition["t", i],
                                tPartition[n+1])))
end proc:

# Takes a list of sets of partitions, returns their product.
PartitionSetListProduct:=proc(L)
    local l, r;

    r:={[]};
    for l in L do
        r:=PartitionSetProduct(l,r)
    end do;
    r
end proc:

# Takes two input sets of decreasingly sorted lists
# (partitions), returns the set of sorted two-by-two
# concatenations of lists.
PartitionSetProduct:=proc(S1, S2)
    local s1, s2;

    {seq(seq(sort([op(s1), op(s2)], `>`), s1=S1), s2=S2)}
end proc:
```

```
# Takes an input patterns set and creates tPartition from it
# which describes the underlying tree.
PatternsToPartition:=proc()
    local i, j;

    tPartition:=table();
    tPartition["size"]:=patternNumber-1;
    for i from 1 to tPartition["size"] do
        tPartition[i]:=[
            seq(op(1,j),
                j=op(1..-1, ExpandTreeOneLevel(t[i])))];
    end do
end proc;

# Translate tPartition to aPartition
PartitionToSystem:=proc()
    local newT, n, i, j, k, b, bb, iSet;

    # For all t that haven't been handled yet
    for n from aPartition["t", "size"]+1
        to tPartition["size"] do
        userinfo(1, treepattern, "--> n="||n||" <--");

        # convert to as
        newT:=OriginalTToTensorsOfA();
        iSet:={};
        # compute intersection with all existing as; skip a_0
        for i to aPartition["a", "size"] while newT <> {} do
            # create a new a if necessary
            b:=aPartition["a", i] intersect newT;
            if b <> {} then
                bb:=aPartition["a", i] minus newT;
                if bb <> {} then
                    j:=AddNewA(bb);
                    b, newT:=SplitSomeA(i, j, b, newT);
                    aPartition["a", i]:=b
                end if;
                iSet:=iSet union {i};
                newT:=newT minus b
            end if
        end do;

        # if there is a remainder after intersecting with
```

```
            # all a's, create a new a for it.
            if newT <> {} then
                k:=AddNewA(newT);
                # Ignore return values.
                SplitSomeA(0, k, {}, {});
                iSet:=iSet union {k};
            end if;
            AddNewT(iSet);
            # print a summary after each step
            Summary()
        end do
    end proc:

    # Recursively label the tree, and fill in the adjacencyTable
    # structure.
    RecursiveLabeling:=proc(tree, parentArg)
        local here, childrenSet, child, parent;
        global Epsilon;

        # do not label leaves
        if tree = Epsilon then
            return
        end if;

        # get current node and children
        here:=op([1,1], tree);
        childrenSet:=op(2, tree);

        parent:=`if`(parentArg=NONE, NULL, parentArg);

        # if there are children, add them all to the
        # adjacency table for the current node; also
        # add the parent node
        if childrenSet <> Epsilon then
            adjacencyTable[here]:=[parent,
                                   op(map2(op, [1, 1],
                                           childrenSet))]
        else
            adjacencyTable[here]:=[parent]
        end if;

        # recursively label all children
        for child in childrenSet do
```

```
        procname(child, here)
    end do
end proc;


# Handle replacement of a[old] by a[old]+a[new] in b and t,
# and return updated b and t.
SplitSomeA:=proc(old, new, b, t)
    local split, i;

    split:=proc(s)
        map(op@PartitionSetListProduct,
            subs({[old]}={[old], [new]},
                 map2(map,x->{[x]},s)))
    end proc;

    userinfo(1, treepattern,
             "splitting "||old||" in "||old||"+"||new);
    for i to aPartition["a", "size"] do
        aPartition["a", i]:=split(aPartition["a", i])
    end do;
    for i to aPartition["t","size"] do
        aPartition["t", i]:=subs(old=(old, new),
                                 aPartition["t", i])
    end do;
    split(b), split(t)
end proc:

# Print out a summary showing the t and a structures.
Summary:=proc()
    local i;

    userinfo(2, treepattern, "Show t-structures:");
    userinfo(2, treepattern, t[0]={$0..aPartition["a",
                                                  "size"]});
    for i to aPartition["t","size"] do
        userinfo(2, treepattern, t[i]=aPartition["t", i])
    end do;
    userinfo(2, treepattern, "Show a-structures:");
    for i to aPartition["a", "size"] do
        userinfo(2, treepattern, a[i]=aPartition["a", i])
    end do
end proc:
```

```
SystemToEquations:=proc()
    local aisize, alist, d, degrees, equations, factors, i,
          j, k, l, m, kterm, kfun, summands, term;

    userinfo(1, treepattern,
             t[0]=add(a[i], i=0..aPartition["a", "size"]));
    for i to aPartition["t","size"] do
        userinfo(1, treepattern,
                 t[i]=add(a[j], j=aPartition["t", i]))
    end do;

    # expect set of lists of type [int, int, ...]
    # create k-Function term for determining powers of u
    kterm:=0;
    for i to nops(newPatternsList) do
        term:=newPatternsList[i];
        userinfo(4, treepattern,
                 cat("creating k-term for ", term));
        aPartition["t", 0]:=0;
        summands:=convert(expand(mul(add(a[k],
                                         k=aPartition["t", j]),
                                 j=term)),
                          list);
        # a_0 will by definition never be part of a pattern
        degrees:=[seq([seq(degree(summands[l], a[j]),
                           j=1..aPartition["a", "size"])],
                      l=1..nops(summands))];
        factors:=add(mul(binomial(L||d, op([l, d], degrees)),
                                d=1..aPartition["a", "size"]),
                     l=1..nops(degrees));

        userinfo(4, treepattern, nops(term));
        userinfo(4, treepattern, factors);
        kterm:=kterm+piecewise(x=1+nops(term), factors, 0);
    end do;

    kfun:=unapply(kterm, x);
    userinfo(3, treepattern, kfun(x));

    equations:={a[0]=1-add(a[i],i=1..aPartition["a",
                                                "size"])};
    # a_0 already handled, so start from 1
```

```
    for i to aPartition["a", "size"] do
        # number of summands in expansion(rhs) of a_i
        aisize:=nops(aPartition["a", i]);
        # list of summands of a_i (each a product of a_j)
        summands:=[seq(mul(a[j], j=op(l, aPartition["a", i])),
                       l=1..aisize)];
        # list of [list of exponents of a_j in lth summand]
        degrees:=[seq([seq(degree(summands[l],a[j]),
                           j=0..aPartition["a", "size"])],
                      l=1..aisize)];
        # list of [product of m_j!]
        factors:=[seq(mul(d!,d=op(l, degrees)), l=1..aisize)];
        # sum of [product of L[j] over m_j]
        # conversion formula to polynomial
        alist[i]:=
            x*add(op(l, summands)/op(l, factors)
            *u^eval(kfun(1+add(m, m=op(l, degrees))),
                           [seq(L||m=op([l,m+1], degrees),
                                m=1..aPartition["a",
                                                "size"])]),
                    l=1..aisize);

        equations:=equations union {a[i]=alist[i]};
    end do;
    equations
end proc:

# Recursively visit all nodes and normalize them.
# Uses adjacencyTable (initialized) and fills visitedTable.
TraverseAndNormalize:=proc(origin, dest)
    local i, T;

    # check if we already visited node dest
    if visitedTable[dest] then
        return NULL
    end if;
    # visit it
    visitedTable[dest]:=true;

    # recursive call on all neighbours
    for i in adjacencyTable[dest] do
        T[i]:=procname(dest, i)
    end do;
```

```
    # return normalized list of all neighbours
    NormalizeTree(seq(T[i], i=adjacencyTable[dest]))
end proc;

# Compare two trees, returns true if first tree is larger
# than second one. Arguments s1 and s2 are some t[i]
# and t[j].
TreeCompare:=proc(s1, s2)
    local t1, t2, i, n1, n2;
    option remember;

    # expand tree one level
    t1:=ExpandTreeOneLevel(s1);
    t2:=ExpandTreeOneLevel(s2);
    n1:=nops(t1);
    n2:=nops(t2);

    # compare number of children
    if n1<n2 then
        return false
    end if;
    if n1>n2 then
        return true
    end if;

    # equal size - compare all children
    for i to min(n1,n2) do
        # need both cases because of possible equality
        if procname(op(i,t1), op(i,t2)) then
            return true
        elif procname(op(i,t2), op(i, t1)) then
            return false
        end if
    end do;

    # equal
    return false
end proc;

# Takes an input tree, returns a new module structure that
# describes the associated pattern set.
TreeToPatterns:=proc(tree)
    local i, j, treeSize;
```

```
    # label the tree
    RecursiveLabeling(tree, NONE);
    treeSize:=nops([indices(adjacencyTable)]);

    visitedTable:=table();
    newPatterns:={};
    for i to treeSize do
        # Select only internal nodes so as to plant on a leaf.
        # (At this stage, "empty" nodes may only be leaves.)
        if nops(adjacencyTable[i])<>1 then
            next
        end if;
        for j to treeSize do
            visitedTable[j]:=false
        end do;
        j:=op(adjacencyTable[i]);
        visitedTable[i]:=true;
        newPatterns:=newPatterns
                    union {TraverseAndNormalize(i,j)}
    end do
end proc;

# User entry point: Convert tree to equations.
TreeToEquations:=proc(tree)
    # initialize
    aPartition:=table([
        ("t","size")=0,
        ("a","size")=0
    ]):
    adjacencyTable:=table();
    assocTable:=table();
    assocTable[0]:=Tree();
    patternNumber:=1;

    # initialize "option remember" functions
    forget(ExpandTree);
    forget(NormalizeTree);
    forget(TreeCompare);

    # initialize 'option remember' table for t[0]
    NormalizeTree():=t[0];
```

```
        TreeToPatterns(tree);
        PatternsToPartition();
        PartitionToSystem();
        NewPatternsToList();
        SystemToEquations();

    end proc;
end module:

savelib('treepattern');
quit
```

Chapter 2

Extended Admissible Functions

In [Hay56], Hayman introduced the notion of an admissible function, which we call Hayman-admissible functions, and proved that the suitably normalized coefficients of admissible functions asymptotically follow a Gaussian distribution. He also assembled a useful list of closure properties, i.e., operations on Hayman-admissible functions that generate other Hayman-admissible functions. However, his result is not directly applicable to counting combinatorial objects.

Our aim was to find a suitable generalization of Hayman's result to two dimensions that warrants a combinatorial interpretation for the coefficients of generating functions, but still provides similar closure properties. We also wanted to make it possible to test membership to this group of functions with MAPLE.

2.1 Introduction

When counting objects with two characteristics (e.g., size n and another parameter k: a_{nk}) one usually expects that there exists the limiting distribution of the random variables X_n defined by

$$P[X_n = k] = \frac{a_{nk}}{a_n}$$

(where $a_n = \sum_k a_{nk}$). In many cases this limiting distribution is Gaussian. More precisely, there exist sequences μ_n and σ_n (with $\sigma_n \to \infty$) with

$$\sum_{k \le \mu_n + x\sigma_n} a_{nk} = a_n \Phi(x) + o(a_n)$$

as $n \to \infty$, where

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-t^2/2}\, dt.$$

The purpose of this paper is to present a concept that allows to decide this question directly by *looking at* the corresponding generating function

$$f(z,u) = \sum_{n,k} a_{nk} z^n u^k.$$

There are good reasons for considering generating functions. First of all, there are lots of (combinatorial) examples where the corresponding (bivariate) generating function is easy to establish. For example the coefficients of the generating function

$$f(z,u) = e^{u(e^z-1)}$$

are the Stirling numbers of the second kind S_{nk} that count the number of partitions of a set of size n into exactly k subsets.

It is also not unrealistic to expect this to work, since the generating function $f(z,u)$ *encodes* certain characteristics of X_n in a proper way. Expected value $\mathbf{E}\, X_n$ and variance $\mathbf{Var}\, X_n$ are given by

$$\mathbf{E}\, X_n = \frac{[z^n] f_u(z,1)}{[z^n] f(z,1)}$$

and

$$\mathbf{Var}\, X_n = \frac{[z^n] f_{uu}(z,1) + f_u(z,1) - (f_u(z,1))^2}{[z^n] f(z,1)},$$

where f_u denotes the derivative of f with respect to u. Furthermore, the moment generating function $m_n(t) = \mathbf{E}\, e^{tX_n}$ is given by

$$m_n(t) = \frac{[z^n] f(z, e^t)}{[z^n] f(z,1)}.$$

There are already numerous results on Gaussian limiting distributions related to the shape of generating functions. For instance, if the coefficients $[z^n]f(z,u)$ behave like a power of a function in u, there are results by Bender and Richmond [BR83b] and Gao and Richmond [GR92]. Extensions are due to Drmota [Drm94b], Gardy [Gar95] (powers of functions), and Hwang [Hwa96, Hwa98] (so-called quasi-powers). Generating functions related to components of combinatorial constructions have been investigated by Flajolet and Soria [FS90, FS93].

We will examine a different approach: For example, if $f(z,1)$, $f_u(z,1)$ and $f_{uu}(z,1)$ are all Hayman-admissible functions (see [Hay56]) for which the asymptotic expansion of the coefficients is known, then it is possible to get an asymptotic relation of $a_n = [z^n]f(z,1)$ and consequently of $\mathbf{E}\, X_n$ and $\mathbf{Var}\, X_n$. One advantage of the class of Hayman-admissible functions is the presence of strong closure properties which are easy to test with MAPLE. The closure properties say that from a basic set of admissible functions (for which one has to verify the conditions for Hayman-admissibility manually) one automatically gets a large class of composite functions which are Hayman-admissible as well.

There have been attempts to extend Hayman's concept: One is due to Harris and Schoenfeld [HS68] in order to get full expansions for the coefficients. Bender and Richmond [BR86, BR96] defined admissibility for functions in several variables and obtained a multidimensional normal law for the coefficients.

However, in order to prove a central limit theorem for the random variables defined above and to establish sufficiently strong closure properties for automatical treatment as well, we need a different concept.

2.2 Extended Admissibility

2.2.1 Hayman-Admissibility

Let us present Hayman's definition of admissibility.

Definition 7. Suppose that $f(z) = \sum_0^\infty a_n z^n$ is regular in $|z| < R$, where $0 < R \leq \infty$ and that for some $R_0 < R$ we have $f(r) > 0, R_0 < r < R$. Let

$$a(r) = r\frac{f'(r)}{f(r)} = \frac{d(\log f(r))}{d(\log r)},$$

$$b(r) = ra'(r) = r\frac{f'(r)}{f(r)} + r^2\frac{f''(r)}{f(r)} - r^2\left(\frac{f'(r)}{f(r)}\right)^2,$$

and $b(r)$ satisfy

$$b(r) \to +\infty, \quad \text{as } r \to R.$$

Suppose further that $f(z)$ satisfies

$$f(re^{i\theta}) \sim f(r)\exp\left(i\theta a(r) - \frac{1}{2}\theta^2 b(r)\right), \quad \text{as } r \to R$$

uniformly for $|\theta| \leq \delta(r)$, while uniformly for $\delta(r) \leq |\theta| \leq \pi$,

$$f(re^{i\theta}) = o\left(\frac{f(r)}{\sqrt{b(r)}}\right), \quad \text{as } r \to R,$$

with some function $\delta(r)$, where $0 < \delta(r) < \pi$. Then $f(z)$ will be said to be *Hayman-admissible* in $|z| < R$. We write: $f(z) \in \mathcal{H}_R$.

2.2.2 Extended Admissibility

The basic definition of extended admissibility is quite similar. The aim is to follow Hayman [Hay56] as closely as possible, which will imply $f(z, 1)$ to be Hayman-admissible, and to require as little as possible for the behavior of f with respect to the second argument. This results in the following definition.

Definition 8. A function $f(z,u) = \sum_{n,k\geq 0} a_{nk} z^n u^k$ is called *extended admissible* or *e-admissible* if there exists $0 < R \leq \infty$ such that the following conditions are satisfied:

1. $f(z,u)$ is analytic in $\Delta_{R,\zeta} := \{(z,u) : |z| < R, |u| < 1+\zeta\}$, where $\zeta > 0$, and for some $R_0 < R$ we have
$$f(r,1) > 0, \quad R_0 < r < R.$$

2. Set
$$a(z,u) = z\frac{f_z(z,u)}{f(z,u)}, \quad \bar{a}(z,u) = u\frac{f_u(z,u)}{f(z,u)},$$
$$b(z,u) = za_z(z,u) = z\frac{f_z(z,u)}{f(z,u)} + z^2\frac{f_{zz}(z,u)}{f(z,u)} - z^2\left(\frac{f_z(z,u)}{f(z,u)}\right)^2,$$
$$\bar{b}(z,u) = u\bar{a}_u(z,u), \quad c(z,u) = ua_u(z,u),$$
and
$$\varepsilon(r) = K\left(\bar{b}(r,1) - \frac{c(r,1)^2}{b(r,1)}\right)^{-1/2},$$
where $K > 0$ is an arbitrary constant. Then, for each choice of $K > 0$ there exists a function $\delta(r) : (R_0, R) \to (0,\pi)$ such that for $R_0 < r < R$ we have
$$f\left(re^{i\theta}, u\right) \sim f(r,u)\exp\left(i\theta a(r,u) - \frac{\theta^2}{2}b(r,u)\right), \quad \text{as } r \to R,$$
uniformly for $|\theta| \leq \delta(r)$ and $u \in [1-\varepsilon(r), 1+\varepsilon(r)]$.

3. For $R_0 < r < R$ we have
$$f\left(re^{i\theta}, u\right) = o\left(\frac{f(r,u)}{\sqrt{b(r,u)}}\right), \quad \text{as } r \to R,$$
uniformly for $\delta(r) \leq |\theta| \leq \pi$ and $u \in [1-\varepsilon(r), 1+\varepsilon(r)]$.

4. For $r \to R$ we have $b(r,1) \to \infty$.

5. $b(r,u) \sim b(r,1)$ for $r \to R$, uniformly for $u \in [1-\varepsilon(r), 1+\varepsilon(r)]$.

6. $a(r,u) = a(r,1) + c(r,1)(u-1) + \mathcal{O}\left(c(r,1)(u-1)^2\right)$ for $r \in (R_0, R)$ and $u \in [1-\varepsilon(r), 1+\varepsilon(r)]$.

7. $\bar{a}(r,u) = \mathcal{O}\left(\bar{a}(r,1)\right)$ and $\bar{b}(r,u) = \mathcal{O}\left(\bar{b}(r,1)\right)$ for all u in an arbitrary but fixed complex neighborhood of 1 and all r.

8. $\bar{b}(r,1) - \frac{c(r,1)^2}{b(r,1)} \to \infty$ as $r \to R$.

9. $\varepsilon(r)^3\bar{b}(r,1) \to 0$ for $r \to R$.

10. $\bar{b}(r,1) = \mathcal{O}\left(\bar{a}(r,1)^2\right)$ and $\bar{a}(r,1) = \mathcal{O}\left(f(r,1)^\eta\right)$ for every $\eta > 0$.

We write: $f(z) \in \mathcal{E}_R$.

Note that this definition implies that functions $f(z,u)$ which are e-admissible in some $\Delta_{R,\zeta}$ are Hayman-admissible in the first variable, i.e.,

$$f(z,u) \in \mathcal{E}_R \implies f(z,1) \in \mathcal{H}_R.$$

This follows from the fact that assumptions (1)–(5) actually define Hayman-admissibility uniformly in u. In particular, this means that the analog of (10) for a and b is true by [Hay56, Lemma 2]. In the bivariate version, this lemma is:

Lemma 2. *Suppose that $f(z,1) \in \mathcal{H}_R$. Then if $n > 0$*

$$\frac{f(r,1)}{r^n} \to +\infty, \quad \text{as } r \to R,$$

$$b(r,1) = o\left(a(r,1)^2\right) \tag{2.1}$$

and given $\eta > 0$, we have

$$a(r,1) = \mathcal{O}\left(f(r,1)^\eta\right) \text{ and } b(r,1) = \mathcal{O}\left(f(r,1)^\eta\right).$$

Furthermore, Hayman's result applies for $f(z,1)$ and because of uniformity in u it carries over to $u \in [1-\varepsilon(r), 1+\varepsilon(r)]$. We get

Theorem 3. *Let $f(z,u) \in \mathcal{E}_R$. Then as $r \to R$ we have*

$$[z^n]f(z,u) = \frac{f(r,u)}{r^n\sqrt{2\pi b(r,u)}}\left(\exp\left(-\frac{(a(r,u)-n)^2}{2b(r,u)}\right) + o(1)\right),$$

uniformly in n and $u \in [1-\varepsilon(r), 1+\varepsilon(r)]$.

Proof. To prove this theorem Hayman's proof can be used without any change. The idea is to write the coefficient as

$$r^n \cdot [z^n]f(z,u) = \frac{1}{2\pi}\int_{-\delta}^{2\pi-\delta} f(re^{i\theta},u)e^{-in\theta}\,d\theta = \frac{1}{2\pi}\left(\int_{-\delta}^{\delta} + \int_{\delta}^{2\pi-\delta}\right) f(re^{i\theta},u)e^{-in\theta}\,d\theta$$

and to use the expansions (2) and (3) of the definition of admissibility to the two integrals of the right-hand side. Since (2) and (3) are uniform in u, the resulting expansion is uniform in u as well. □

As a consequence of this theorem we can prove

Theorem 4. *Let $f(z,u) \in \mathcal{E}_R$ such that for sufficiently large n all coefficients a_{nk} are nonnegative. X_n denotes a sequence of random variables satisfying*

$$P[X_n = k] = \frac{a_{nk}}{a_n}.$$

Then the following central limit theorem holds:

$$Y_n := \frac{X_n - \bar{a}(r_n,1)}{\sqrt{|B(r_n,1)|/b(r_n,1)}} \to \mathcal{N}(0,1), \tag{2.2}$$

where r_n is the positive solution of $a(r,1) = n$ and $|B|$ is the determinant of B with

$$B(r,u) = \begin{pmatrix} b(r,u) & c(r,u) \\ c(r,u) & \bar{b}(r,u) \end{pmatrix}.$$

Furthermore we have, as $n \to \infty$,

$$\mathbf{E}\, X_n = \bar{a}(r_n,1) + o\left(|B(r_n,1)|/b(r_n,1)\right) \tag{2.3}$$

and

$$\mathbf{Var}\, X_n \sim \frac{|B(r_n,1)|}{b(r_n,1)}. \tag{2.4}$$

Note that (2.2), (2.3), and (2.4) also show that

$$\frac{X_n - \mathbf{E}\, X_n}{\sqrt{\mathbf{Var}\, X_n}} \to \mathcal{N}(0,1).$$

Proof. In order to prove the theorem, we have to show that the moment generating function of X_n,

$$m_n(t) = \mathbf{E}e^{tX_n} = \frac{[z^n]f(r,e^t)}{[z^n]f(r,1)},$$

satisfies

$$\mathbf{E}e^{tY_n} = e^{-t\bar{a}(r,1)} m_n\left(\frac{t}{\sqrt{b(r,1) - c(r,1)^2/b(r,1)}}\right) \to e^{t^2/2}$$

for all $t \in [-K,K]$ with some $K > 0$. [Hay56, Lemma 4] tells us that, if $|\bar{b}(r,u)| < \bar{b}(r,1)$ for $|u-1| < 2\eta$ (which is true by (7)), then

$$f(r,e^t) = f(r,1)\exp\left(t\bar{a}(r,1) + \frac{t^2}{2}\bar{b}(r,1) + \mathcal{O}\left(\bar{b}(r,1)t^3\right)\right), \qquad \text{as } r \to R,$$

uniformly for $|t| < \varepsilon(r)$. Thus by (6) we obtain

$$\begin{aligned} m_n(t) =& \frac{f(r,1)(1+\mathcal{O}(t)}{r\sqrt{2\pi b(r,1)}} \times \\ & \times \exp\left(-\frac{(a(r,e^t)-a(r,1))^2}{2b(r,1)}(1+\mathcal{O}(t)) + t\bar{a}(r,1) + \frac{t^2}{2}\bar{b}(r,1) + o(1)\right) \\ =& \frac{f(r,1)(1+\mathcal{O}(t)}{r\sqrt{2\pi b(r,1)}} \times \\ & \times \exp\left(t\bar{a}(r,1) + \frac{t^2}{2}\left(\bar{b}(r,1) - \frac{c(r,1)^2}{b(r,1)}\right) + \mathcal{O}\left(\frac{c(r,1)^2}{b(r,1)}t^3\right) + o(1)\right). \end{aligned}$$

Since $c(r,1)^2 < b(r,1)\bar{b}(r,1)$ by (8) an application of (9) finally establishes the central limit theorem (2.2).

The proof of (2.3) and (2.4) is standard. We can even show that all moments of the normalized random variable converge to the corresponding moments of the Gaussian distribution. Convergence of the moment generating function and Chernov's inequality provide exponential tail estimates for the distribution functions. Thus, it is sufficient to consider finite intervals for proving convergence of moments. However, on finite intervals monomials are bounded and, thus, (uniform) convergence of distribution functions implies convergence of bounded functionals. This completes the proof. □

2.2.3 Closure Conditions

The proofs of theorems 3 and 4 do not require all parts of the definition of e-admissibility. For theorem 3 assumptions (1)–(4) are sufficient. The assertion of theorem 4 follows from assumptions (1)–(6), (8), and (9). However, all assumptions are needed to establish the closure properties for e-admissible functions. Those assumptions are usually satisfied for *reasonably constructed functions*, compare with Theorems 5 and 6.

Theorem 5. *The following two classes of functions are extended admissible:*

- *Let $P(z,u)$ be a polynomial in z and u with real coefficients written in the form $P(z,u) = \sum_n p_n z^{k_n} u^{l_n}$ by choosing an arbitrary order of the monomials. Furthermore, let $P(z,1) = \sum_m b_m z^m$, i.e., $b_m = \sum_{n:k_n=m} p_n$. Finally, set*

$$K := \max E \text{ with } E = \left\{k_i + k_j : \det\begin{pmatrix} k_i & l_i \\ k_j & l_j \end{pmatrix} \neq 0\right\}$$

and $I := \{(i,j) : k_i + k_j = K\}$. Then $e^{P(z,u)} \in \mathcal{E}_\infty$ if and only if the following conditions are satisfied:

  (a) For every $d > 1$ there exists an $m \not\equiv 0 \mod d$ such that $b_m \neq 0$. Moreover, for $m_d = \max\{m \not\equiv 0 \mod d : b_m \neq 0\}$ we have $b_{m_d} > 0$.

(b) E is not empty and

$$\sum_{(\mu,\nu)\in I} p_\mu p_\nu \det \begin{pmatrix} k_\mu & l_\mu \\ k_\nu & l_\nu \end{pmatrix}^2 > 0.$$

(c) $\max\{k_j : p_j \neq 0\} < 3K/5$

- *If* $f(z) \in \mathcal{H}_R$, $g(u)$ *is analytic for* $|u| \leq 1+\zeta$ *and satisfies* $g(1) > 0$ *as well as* $g'(1) + g''(1) - \frac{g'(1)^2}{g(1)} > 0$, *then* $e^{g(u)f(z)} \in \mathcal{E}_R$.

Proof.

Part 1

Let $P(z,u) = \sum_{n=1}^{L} p_n z^{k_n} u^{l_n}$ be a polynomial satisfying the assumptions of the theorem. Since $P(r,u) \sim P(r,1)$, uniformly in u, and the statement of the theorem is true for Hayman-admissibility and polynomials in one variable (see [Hay56, Theorem X]) satisfying condition (a), we immediately get (2) and (3) with $a(r,u) = \sum_n p_n k_n r^{k_n} u^{l_n}$ and $b(r,u) = \sum_n p_n k_n^2 r^{k_n} u^{l_n}$ if we choose δ such that $\delta(r) = o\left(r^{-k/3}\right)$, where $k = \max k_n$. Furthermore, (a) implies that the leading coefficient in $b(z,1)$ is positive which implies (4). Now (5) and (6) are obviously satisfied with $c(r,u) = \sum_n p_n k_n l_n r^{k_n} u^{l_n}$. Since $\bar{a}(r,u) = \sum_n p_n l_n r^{k_n} u^{l_n}$ and $\bar{b}(r,u) = \sum_n p_n l_n^2 r^{k_n} u^{l_n}$ we immediately get (7). To show (8) let B as in Theorem 4 denote the matrix of the second logarithmic derivatives of $P(z,u)$. Then by the multilinearity of the determinant we get

$$\det B(r,1) = \sum_{h=1}^{L}\sum_{i=1}^{L} p_h p_i r^{k_h+k_i} \left(\det \begin{pmatrix} k_h & l_h \\ k_i & l_i \end{pmatrix}\right)^2$$

Then by condition (b) the order of magnitude of B is $B = \Theta\left(r^K\right)$ and the leading term is positive which implies (8). From $b(r,1) = \Theta\left(r^k\right)$ we obtain $\varepsilon = \Theta\left(r^{(K-k)/2}\right)$ and by condition (c) this proves (9). Finally, note that f grows exponentially while $\bar{a}$ and $\bar{b}$ do not. Moreover, $\bar{a}$ and $\bar{b}$ have the same order of magnitude and thus we have (10).

On the other hand, if the first condition is violated, then $e^{P(z,1)}$ will not be Hayman-admissible and hence $e^{P(z,u)} \notin \mathcal{E}$. (8) and (9) are equivalent to (b) and (c), respectively. Therefore the three conditions in the theorem are necessary and sufficient for admissibility.

Part 2

Let $F(z,u) = e^{g(u)f(z)}$ with $f \in \mathcal{H}_R$ and $g(u)$ as in the statement of the theorem. By [Hay56, Lemma 5] we have uniformly for $|\theta| \leq a(r)^{-1}$

$$f(re^{i\theta}) = f(r) + i\theta r f'(r) - \frac{\theta^2}{2}(rf'(r) + r^2 f''(r)) + \mathcal{O}\left(\theta^3 f(r) a(r)^3\right). \qquad (2.5)$$

We can directly compute

$$\begin{aligned} a(r,u) &= rg(u)f'(r), & \bar{a}(r,u) &= ug'(u)f(r), \\ b(r,u) &= (rf'(r)+r^2f''(r))g(u), & \bar{b}(r,u) &= (ug'(u)+ug''(u))f(r), \text{and} \\ c(r,u) &= rug'(u)f'(r). \end{aligned}$$

With these formulas the validity of (4)–(7) is easily seen. For (8) note that by [Hay56, Th. III] we have

$$f^{(k)}(r) \sim f(r)\left(\frac{a(r)}{r}\right)^k$$

and thus we get

$$\begin{aligned} \bar{b} - \frac{c^2}{b} &= (g'(1)+g''(1))f(r) - \frac{r^2g'(1)^2f'(r)^2}{(rf'(r)+r^2f''(r))g(1)} \sim \\ &\sim \left(g'(1)+g''(1)-\frac{g'(1)^2)}{g(1)}\right) f(r) \to \infty. \end{aligned}$$

Consequently, $\varepsilon(r)^3\bar{b}(r,1) \asymp f(r,1)^{-1/2} \to 0$, as required in (9). As to the exponential growth of f, (10) is obvious.

Finally, following Hayman [Hay56], we set $\delta(r) = f(r)^{-2/5}$ and (2.5) directly yields (2). By [Hay56, Lemma 6] we have

$$|f(re^{i\theta})| \le f(r) - f(r)^{1/7}$$

and hence

$$|F(re^{i\theta},u)| \le F(r,u)e^{-g(u)f(r)^{1/7}}.$$

Now applying [Hay56, Th. III and Lemma 2] gives $B(r,u) = \mathcal{O}\left(g(u)f(r)^{1+\varepsilon}\right)$ and (3) follows. □

Note that for functions of the form $e^{P(z,u)}$ the concept of e-admissibility is too strict. In fact, there are many cases where a normal limit law occurs though the corresponding generating function is not e-admissible (compare with the examples presented in section 2.3). The reason lies in condition (c) which is related to (9) of the definition. Since $P(z,u)$ are *nice functions* not only for u close to one, this condition can be avoided and a multivariate normal limit law can be proved. A characterization of these cases is the topic of work in progress.

Theorem 6. *Suppose that $f(z,u) \in \mathcal{E}_R$ and $g(z,u) \in \mathcal{E}_R$, $P(z,u)$ a polynomial with positive coefficients, and $h(z) \in \mathcal{H}_R$. Then the following functions are in $\mathcal{E}_R$, too:*

- $f(z,u) \cdot g(z,u)$

- $h(z) \cdot f(z,u)$
- $P(z,u) \cdot f(z,u)$,
- $e^{f(z,u)}$
- $e^{P(z,u)h(z)}$, *if* $P(z,u)$ *is not independent of* u.
- $e^{P(z,u)+h(z)}$, *if* $R = \infty$ *and* $P(z,u)$ *is not independent of* u.
- $f(z,u) + Q(z,u)$, *where* $Q(z,u)$ *is an arbitrary polynomial.*

Proof.

- **Product of admissible functions**

  Let $f_1, f_2 \in \mathcal{E}_R$. Clearly, the logarithmic derivatives of $f = f_1 f_2$ satisfy

  $$\begin{array}{lll} a = a_1 + a_2 & b = b_1 + b_2 & c = c_1 + c_2 \\ \bar{a} = \bar{a}_1 + \bar{a}_2 & \bar{b} = \bar{b}_1 + \bar{b}_2 & \end{array}$$

  Thus (5), (6), and (4) are fulfilled for $u \in [1-\eta, 1+\eta]$ with $\eta = \min(\varepsilon_1, \varepsilon_2)$. The validity of (7) is easy to check as well. To prove (8) observe that

  $$\begin{aligned} & \bar{b} - \frac{c^2}{b} \geq \bar{b}_1 - \frac{c_1^2}{b_1} + \bar{b}_2 - \frac{c_2^2}{b_2} \\ \iff & \frac{(c_1+c_2)^2}{b_1+b_1} \leq \frac{c_1^2}{b_1} + \frac{c_2^2}{b_2} \\ \iff & 2c_1c_2b_1b_2 \leq c_1^2b_2^2 + c_2^2b_1^2 \end{aligned}$$

  which is obviously true and implies (8). Another consequence is that $\varepsilon = 1/\sqrt{\bar{b} - c^2/b} \leq \eta$, hence the domain of validity for (5), (6), and (4) is large enough. Furthermore, this implies (9). (2) and (3) can be proven in the same way as [Hay56, Th. VII] and (10) is obvious. □

- **Multiplication by a Hayman-admissible function**

  If $f_1 \in \mathcal{H}_R$ and $f_2 \in \mathcal{E}_R$, then the same argument as in the previous section applies. Of course, here we have $c_1 = \bar{a}_1 = \bar{b}_1 = 0$ and $\varepsilon = \varepsilon_2$. □

- **Multiplication by a polynomial**

  Let $f(z,u) \in \mathcal{E}_R$ with some positive $R \leq \infty$ and $P(z,u)$ a polynomial with positive coefficients. Due to uniformity in u the proof of [Hay56, Th. VIII]

can be used without change to prove (2) and (3). Let $A, B, C, \bar{A}, \bar{B}$ denote the logarithmic derivatives corresponding to $P(z,u)f(z,u)$ then we have

$$A(r,u) = a(r,u) + r\frac{P_z(r,u)}{P(r,u)}$$
$$\bar{A}(r,u) = \bar{a}(r,u) + u\frac{P_u(r,u)}{P(r,u)}$$
$$B(r,u) = b(r,u) + r\frac{P_z(r,u)}{P(r,u)} + r^2\left(\frac{P_{zz}(r,u)}{P(r,u)} - \frac{P_z(r,u)^2}{P(r,u)^2}\right)$$
$$\bar{B}(r,u) = \bar{b}(r,u) + u\frac{P_u(r,u)}{P(r,u)} + u^2\left(\frac{P_{uu}(r,u)}{P(r,u)} - \frac{P_u(r,u)^2}{P(r,u)^2}\right)$$
$$C(r,u) = c(r,u) + ru\left(\frac{P_{zu}(r,u)}{P(r,u)} - \frac{P_z(r,u)P_u(r,u)}{P(r,u)^2}\right)$$

Since the rational terms in the equations above remain bounded when $r \to R$, contrary to a, $\bar{a}$, b, and $\bar{b}$, they do not affect the validity of (4)–(10). □

- **Exponential of admissible functions**

Let $F(z,u) = e^{f(z,u)}$. The logarithmic derivatives of f are denoted as usual by $a, \bar{a}, b, \bar{b}$, and c, the ones of F by the corresponding capital letters.

Since extended admissible functions satisfy all the conditions imposed on Hayman-admissible functions even uniformly in u, Lemmas 1–6 of Hayman [Hay56] are true for extended admissible functions, too. The proofs there can be used without any change except replacing $f(z)$ by $f(z,u)$ and derivatives by partial derivatives with respect to z, and the results hold uniformly in u. Since with the help of these lemmas Hayman proved that the exponential of an Hayman-admissible function is Hayman-admissible as well, we can adopt his proof of (2) and (3). Moreover, we get

$$B(r,1) \to \infty. \tag{2.6}$$

Note that

$$A(r,u) = rf_z(r,u) = a(r,u)f(r,u)$$
$$\bar{A}(r,u) = uf_u(r,u) = \bar{a}(r,u)f(r,u) \tag{2.7}$$
$$B(z,u) = rf_z(r,u) + r^2f_{zz}(r,u) = (b(r,u) + a^2(r,u))f(r,u)$$
$$\bar{B}(z,u) = uf_u(r,u) + u^2f_{uu}(r,u) = (\bar{b}(r,u) + \bar{a}^2(r,u))f(r,u) \tag{2.8}$$
$$C(z,u) = zuf_{zu}(r,u) = (c(r,u) + a(r,u)\bar{a}(r,u))f(r,u)$$

With these formula we see that (10) is valid, since for every $\gamma, \eta > 0$ we have $\bar{a}f = \mathcal{O}\left(f^{1+\gamma}\right) = \mathcal{O}\left(e^{\eta f}\right)$ and $(\bar{b} + \bar{a}^2)f = o\left(\bar{a}^2f^2\right)$. Moreover, we get

$$\bar{B} - \frac{C^2}{B} = \left(\bar{b} + \bar{a}^2 - \frac{(c + a\bar{a})^2}{b + a^2}\right)f \geq \frac{b\bar{b} - c^2}{b + a^2}f \to \infty, \tag{2.9}$$

where the inequality is obtained by expanding the terms, using $c = o(a\bar{a})$ which is a consequence of (8), (2.1), and (10), and performing some elementary estimates. The limit follows from (8) and (2). Furthermore, observe that since $|u-1| \leq \sqrt{b+a^2}/\sqrt{(b\bar{b}-c^2)\,f}$ we obtain by (10) and (2)

$$\begin{aligned} |(u-1)\bar{a}| &\leq \frac{\bar{a}\sqrt{b+a^2}}{\sqrt{(b\bar{b}-c^2)\,f}} \to 0, \\ |(u-1)^2\bar{b}| &\leq \frac{\bar{b}(b+a^2)}{(b\bar{b}-c^2)\,f} \to 0. \end{aligned} \tag{2.10}$$

Hence we immediately get (9):

$$\varepsilon^3 \bar{B} = \frac{(\bar{b}+\bar{a}^2)(b+a^2)^{3/2}}{(b\bar{b}-c^2)^{3/2}\sqrt{f}} \to 0. \tag{2.11}$$

Turning to (5) observe

$$\begin{aligned} \frac{B(r,u)}{B(r,1)} &= \frac{b(r,u)+a(r,u)^2}{b(r,1)+a(r,1)^2} \\ &= \frac{b(r,1)+o\left(b(r,1)\right)+a(r,1)^2}{b(r,1)+a(r,1)^2} + \\ &+ \frac{\mathcal{O}\left(2a(r,1)c(r,1)(u-1)+(a(r,1)c(r,1)+c(r,1)^2)(u-1)^2\right)}{b(r,1)+a(r,1)^2} \\ &= 1+o(1), \end{aligned}$$

where the last equation follows from application of (10) and (2): we have

$$\begin{aligned} \frac{ac}{b+a^2}(u-1) &\leq \frac{ac}{\sqrt{b+a^2}\sqrt{(b\bar{b}-c^2)\,f}} \quad \text{by (2.9)} \\ &\leq \frac{b+\sqrt{b\bar{b}}}{a\sqrt{(b\bar{b}-c^2)\,f}} \to 0, \end{aligned}$$

by (10) along with $c \leq \sqrt{b\bar{b}}$ and then (10) together with (2), and

$$\frac{ac+c^2}{b+a^2}(u-1)^2 \leq \frac{a\sqrt{b\bar{b}}+b\bar{b}}{(b\bar{b}-c^2)\,f} \to 0$$

by (2.9) together with (10) and (10) together with (2). This implies (5) and in conjunction with (2.6) we get (4). Next we prove (6). Set $u = e^t$.

Then we have

$$\begin{aligned}\frac{A(r,u)-A(r,1)}{C(r,1)(u-1)} &= \frac{\frac{f(r,u)}{f(r,1)}a(r,u)-a(r,1)}{(c(r,1)+a(r,1)\bar{a}(r,1))(u-1)}\\ &= \frac{\left(\frac{f(r,e^t)}{f(r,1)}-1\right)a(r,1)+\frac{f(r,e^t)}{f(r,1)}c(r,1)t+\mathcal{O}\left(\frac{f(r,e^t)}{f(r,1)}c(r,1)t^2\right)}{t(c(r,1)+a(r,1)\bar{a}(r,1))}\\ &= \frac{\left(e^{t\bar{a}-t^2\bar{b}}-1\right)a+e^{t\bar{a}-t^2\bar{b}}ct+\mathcal{O}\left(e^{t\bar{a}-t^2\bar{b}}ct^2\right)}{t(c+a\bar{a})}\\ &= 1+\mathcal{O}(t)\end{aligned}$$

where the last two lines follow from (2.11) and (2.10). Finally, (7) is obvious in view of (2.7) and (2.8). □

- **The class** $exp(P(z,u)h(z))$

  Since e-admissible functions are closed under multiplication, it is sufficient to consider the case where $P(z,u)$ is a monomial. Then use the fact that the product of an Hayman-admissible function with a polynomial is again Hayman-admissible and combine this with the second part of Theorem 5 to get the result.

- **The class** $exp(P(z,u)+h(z))$

  Here we have

$$\begin{aligned}a(z,u) &= z(P_z(z,u)+h'(z)) & \bar{a}(z,u) &= uP_u(z,u)\\ b(z,u) &= z(P_z(z,u)+h'(z))+z^2(P_{zz}(z,u)+h''(z)) & &\\ \bar{b}(z,u) &= uP_u(z,u)+u^2P_{uu}(z,u) & c(z,u) &= zuP_{zu}(z,u)\end{aligned}$$

  and hence

$$\begin{aligned}|B(z,u)| &= \begin{vmatrix} zP_z+z^2P_{zz} & zuP_{zu}\\ zuP_{zu} & uP_u+u^2P_{uu}\end{vmatrix}+\begin{vmatrix} zh'+z^2h'' & zuP_{zu}\\ 0 & uP_u+u^2P_{uu}\end{vmatrix}\\ &= D_1(z,u)+D_2(z,u).\end{aligned}$$

  The growth properties of Hayman-admissible functions and their derivatives are well studied (see [Hay56]). Therefore we know the order of magnitude of $D_1(r,1)$ and $D_2(r,1)$: We have $D_2(r,1)=\Theta(r^2h''(r)(P_u(r,1)+P_{uu}(r,1))$ and $D_1=o(D_2)$, as $r\to\infty$. Moreover, since $\bar{b}(r,1)$ and $c(r,1)$ grows polynomially and $b(r,1)\sim r^2h''(r)$ we get (8). The polynomial growth of $\bar{a}$ and $\bar{b}$ along with their equal order of magnitude immediately implies (10). Since $\varepsilon\sim\bar{b}^{-1/2}$, we have on one hand (9) and on the other hand $\varepsilon\to 0$ which implies (2) and (4)–(7). Finally, since

$$\exp\left(P\left(re^{i\theta},u\right)\right)=\mathcal{O}\left(e^{P(r,u)}\right)\text{ and }e^h\left(re^{i\theta}\right)=o\left(\frac{e^{h(r)}}{\sqrt{b(r,1)}}\right)$$

we get (3). □

- **Addition of Polynomials**

  Since every e-admissible function $f(z,u)$ grows at least exponentially, polynomials are negligibly small in comparison to $f(z,u)$ and thus $f(z,u) + P(z,u)$ is obviously e-admissible, too.

□

For $e^{P(z,u)+h(z)}$ the positivity condition for $P(z,u)$ can be relaxed. For such a function to be e-admissible it is sufficient that $\lim_{z\to\infty} P_u(z,1) + P_{uu}(z,1) = \infty$.

With help of Theorems 4–6 it is also possible to check automatically (with MAPLE) if a given function $f(z,u)$ is in $\mathcal{E}_R$ and in most cases one obtains a central limit theorem for the coefficients as well.

Theorem 4 provides asymptotic expansions for expected value and variance in terms of the derivatives of $f(z,u)$ (evaluated at $z = r_n$ and $u = 1$), too. Thus, it is also of interest to obtain these asymptotic expansions automatically. For this purpose one has to solve two problems, first an asymptotic expansion for r_n (that is, an asymptotic inversion of the function $r \mapsto a(r,1) = rf_z(r,1)/f(r,1)$) and, second, an asymptotic insertion of r_n into $\bar{a}(r,1)$ and into $|B(r,1)|/b(r,1)$. For example, if f is an exp-log function (that is, it is built by finitely many compositions of rational functions as well as e^z and $\log z$) then we can apply the results and implementations for multiseries inversion and substitution by Salvy and Shackell [SS99] (cf. also Richardson et al. [RSSVdH96]) in order to automatically compute an asymptotic expression for r_n and consequently for mean and variance, if they exist. (The problem is that not every exp-log function has an inverse which is asymptotically equal to an exp-log function, see Shackell [Sha93]).

2.3 Examples

In this section we present some combinatorial applications for e-admissibility. Note that many generating functions occurring in the following examples more than satisfy the conditions for being e-admissible. For instance, examples 2.3.1, 2.3.3, and 2.3.5-3 are admissible in the sense of Bender and Richmond [BR96] as well. However, there are combinatorial problems whose generating function is not Bender-Richmond-admissible, see example 2.3.4.

2.3.1 Stirling Numbers of the Second Kind

The generating function for the Stirling numbers of the second kind is $f(z,u) = e^{u(e^z-1)}$. Since $e^z - 1$ is Hayman-admissible, $f(z,u)$ is e-admissible by the second part of Theorem 5. We get

$$a(z,u) = uze^z, \quad \bar{a}(z,u) = u\,(e^z - 1)\,,$$

and

$$B(z,u) = \begin{pmatrix} u\,(z+z^2)\,e^z & uze^z \\ uze^z & u\,(e^z-1) \end{pmatrix}.$$

The solution of $a(r,1) = re^r = n$ is $r_n \sim \log n - \log\log n$. Hence we get a normal limit law with asymptotic mean $\bar{a}(r_n,1) \sim \frac{n}{\log n}$ and asymptotic variance

$$\frac{\det B(r_n,1)}{b(r_n,1)} = e^{r_n} - 1 - \frac{(r_n e^{r_n})^2}{(r_n^2+r_n)e^{r_n}} \sim \frac{e^{r_n}}{r_n} \sim \frac{n}{\log^2 n}$$

which has already been shown by Harper [Har67].

2.3.2 Permutations with Bounded Cycle Length

Permutations with cycle length less than or equal ℓ can be described by the generating function $f(z,u) = e^{ue_\ell(z)}$, where $e_\ell(z) = \sum_{i=1}^{\ell} \frac{z^i}{i}$. Since the exponent is a polynomial, we have to check the conditions of Theorem 5 and it turns out that $f(z,u)$ is e-admissible if and only if $\ell > 3$. So in this case Theorem 4 implies a central limit law. To get asymptotic mean and variance compute

$$a(z,u) = uz\frac{1-z^\ell}{1-z}, \qquad \bar{a}(z,u) = u\sum_{i=1}^{\ell}\frac{z^i}{i}, \quad c(z,u) = a(z,u),$$
$$b(z,u) = uz\frac{1-(\ell+1)z^\ell+\ell z^{\ell+1}}{(1-z)^2}, \quad \bar{b}(z,u) = \bar{a}(z,u).$$

$a(r,1) = n$ implies $r_n \sim n^{1/\ell}$ and consequently asymptotic mean and variance are given by

$$\bar{a}(r_n,1) \sim \frac{n}{\ell} \quad \text{and} \quad \frac{\det B(r_n,1)}{b(r_n,1)} \sim \frac{2\ell^2-2\ell+1}{\ell^2(\ell-1)} n^{1-\frac{1}{\ell}},$$

respectively.

Remark. Though for $\ell \leq 3$ the function is not e-admissible any more (ε would be too large in this cases), it can be shown (see [GM04]) that the central limit law with asymptotic mean and variance as given above is true for $\ell \geq 2$. In the case $\ell = 1$ the distribution degenerates. (The same holds for the first two examples in section 2.3.5.)

2.3.3 Partitions of a Set of Partitions

These objects are the partition of the set of subsets of a given partition. One gets the generating function $f(z,u) = e^{u(e^{\exp(z)-1}-1)}$. Again, one can compute the

logarithmic derivatives and an asymptotic expression for r_n, mean, and variance. This has been done by Salvy and Shackell [SS99] and they get

$$\bar{a}(r_n, 1) \sim \frac{n}{\log n \log\log n} \quad \text{and} \quad \frac{\det B(r_n, 1)}{b(r_n, 1)} \sim \frac{n}{\log^2 n \log\log n}.$$

Since $e^{e^z-1} - 1$ is Hayman-admissible and thus $f(z, u)$ e-admissible, Theorem 4 implies that moreover a central limit theorem holds.

2.3.4 Partitions Counted by Singleton Blocks

When counting the number of partitions of an n-element set having k singleton blocks, we get the generating function $f(z, u) = e^{e^z-1-z+zu}$. The exponent is the sum of a polynomial and $e^z - 1 - z$, an Hayman-admissible function. Thus by Theorem 6 $f(z, u)$ is e-admissible and therefore a central limit theorem holds. We get

$$a(z, u) = ze^z - z + uz, \quad \bar{a}(z, u) = uz$$

and

$$B(z, 1) = \begin{pmatrix} (z^2 + z)\, e^z & z \\ z & z \end{pmatrix}.$$

Thus by $a(r, 1) = re^r = n$ we have as before $r_n \sim \log n - \log\log n$ and therefore the asymptotic mean and variance are given by $\bar{a}(r_n, 1) \sim \log n$ and

$$\frac{\det B(r_n, 1)}{b(r_n, 1)} = \frac{(r_n^3 + r_n^2)e^{r_n} - r_n^2}{r_n(r_n + 1)e^{r_n}} \sim r_n \sim \log n,$$

respectively.

2.3.5 Other Examples

1. Set partitions with bounded block size

   The generating function is $\exp\left(u \sum_{i=1}^{\ell} \frac{z^i}{i!}\right)$ which is obviously e-admissible if and only if $\ell > 3$.

   Of course, different kinds of restrictions on the block size may be investigated by our method as well, since they differ from the present case only in the sum in the exponent. The same applies to restrictions of the cycle length in permutations (cf. Examples 2.3.2).

2. Number of cycles of maximal allowed length in restricted permutations

   Here the generating function is $\exp\left(\frac{uz^\ell}{\ell} + \sum_{i=1}^{\ell-1} \frac{z^i}{i}\right)$ which is e-admissible for $\ell > 3$.

3. Covering complete graphs with bipartite graphs

   The generating function is $\exp\left(u(e^z-1)^2/2\right)$ (see [GJ83, BR96]) which is obviously e-admissible.

2.4 MAPLE Source Code

We provide a MAPLE program of an algorithm that checks if a function in two variables is e-admissible by using the closure properties from theorems 5 and 6.

The implementation provides a MAPLE module named *extadm* with two user entry points: *extadmtest* and *ea_addfunc*.

extadmtest(fct::algebraic, var1::name, var2::name) tests if a given function is extended admissible in *var1* around *var2* = 1 and returns true, false, or an error if it cannot decide.

A user can teach *extadmtest* about additional classes of functions with the *ea_addfunc* function.

The algolib library available at `http://algo.inria.fr/libraries/` has to be installed before using the code below, since it is used for Hayman-admissibility tests.

2.4.1 Usage

A sample session testing the example from section 2.3.2 (where e-admissibility is only true for $l > 3$) might run as follows:

```
# maple8
    |\^/|     Maple 8 (IBM INTEL LINUX)
._|\|   |/|_. Copyright (c) 2002 by Waterloo Maple Inc.
 \  MAPLE  /  All rights reserved. Maple is a registered trademark of
 <____ ____>  Waterloo Maple Inc.
      |       Type ? for help.
> with(extadm);
                          [ea_addfunc, extadmtest]

> el2:=(z,l)->add(z^i/i,i=1..l);
                                          i
                                         z
                  el2 := (z, l) -> add(----, i = 1 .. l)
                                         i

> extadmtest(exp(u*el2(z,2)),z,u);
                                  false
```

```
> extadmtest(exp(u*el2(z,3)),z,u);
                                        false

> extadmtest(exp(u*el2(z,4)),z,u);
                                        true

> extadmtest(exp(u*el2(z,10)),z,u);
                                        true

>
```

Additional information will be printed when the variable *infolevel[extadm]* is set to at least 3.

A user can teach *extadmtest* about additional classes of functions. For this, the user has to provide a function *impl* which takes three arguments *fct*, *var1*, and *var2*, like *extadmtest* itself, and returns true, false, or error depending on the e-admissibility of the tested function. This function *impl* is then added to *extadmtest*'s function table by calling *ea_addfunc(fname::symbol, impl)*. After a function handler for *fname* has been added in this way, whenever *fname* appears in a test, *extadmtest* calls *impl(fct, var1, var2)* on the subfunction.

For example: *ea_addfunc('sinh', ea_sinh)* would add the user-provided function *ea_sinh* as decision function for any hyperbolic sine (sub-)functions.

2.4.2 Implementation

extadmtest(fct::algebraic, var1::name, var2::name) recursively splits the given function *fct* into its components and uses the closure properties from theorems 5 and 6 to determine if the function is e-admissible.

The source code is split up in two files, `polcheck.maple` and `extadm.maple`.

`polcheck.maple` contains *ea_polcheck*, a function that tests if a polynomial has the form described in theorem 5 and returns true if it has and false if it has not. In detail, it does the following:

- Check if input is a polynomial.
- Create a list of the b_m.
- Create a list of all pairs of exponents (k_n, l_n) and the corresponding coefficients.
- Verify that the greatest common divisor of the indizes m of $b_m \neq 0$ is not 1. (This is equivalent to the first part of condition (a) of theorem 5.)
- Verify that the second part of condition (a) is fulfilled.

- Compute E and verify it is non-empty.
- Compute K.
- Compute I.
- Verify that the product from condition (b) is greater than zero.
- Verify condition (c).

Here is the MAPLE code for `polcheck.maple`:

```
# -*- maple -*-
#
# $Id: polcheck.maple,v 1.20 2004/05/13 22:28:39 wiz Exp $
#
# Input: polynomial in two variables with positive coefficients
#        fct ... polynomial in var1 and var2 = sum p_n z^k_n u^l_n
#        var1 .. z
#        var2 .. u
#
# Output: true if e^pol is extended admissible, false if not
#
# Algorithm:
#  1. Check input validity
#  2. Create list of all pairs (k_n, l_n) of exponents
#  3. Create list of b_m = sum_{k_n=m} p_n
#  4. (a) Verify gcd (degrees of b_m z^m) = 1
#  5. (a) For all d>1,
#         m_d = max {m not congruent 0 mod d: b_m <> 0}
#         exists and b_{m_d} > 0
#  6. (b) Find E = { k_i+k_j | det (k_i l_i \ k_j l_j) <> 0} and
#         verify it's non-empty
#  7. K = max E
#  8. Compute I = { (i,j) | k_i+k_j = K }
#  9. (b) Verify
#         sum_{(i,j) in I} p_i p_j det(k_i l_i \ k_j l_j)^2 > 0
# 10. (c) Verify max { k_j | p_j <> 0 } < 3K/5
# User interface; does step 1, 2
ea_polcheck := proc(fct::algebraic, var1::name, var2::name)
    local all_coeffs, monomials, bm_coeffs, bm_monomials;

    # check if input is a polynomial
    if not type(fct, polynom(anything, {var1, var2}))
    then
```

```
        error "not a polynomial: %1", fct
    end if;

#  3. Create list of b_m = sum_{k_n=m} p_n
    bm_coeffs := coeffs(combine(expand(subs(var2=1,fct))),
        [var1], 'bm_monomials');
    # step 2
    all_coeffs := coeffs(expand(fct), [var1,var2], 'monomials');
    check_condition([all_coeffs], map(degree, [monomials], var1),
        map(degree, [monomials], var2),
        [bm_coeffs], map(degree, [bm_monomials], var1));
end proc;

check_condition:=proc(all_coeffs, deg1, deg2, bm_coeffs, bm_deg1)
    local comparelist, d, i, isum, j, K, maximum, msubd,
          msubdcoeff, msubdpos, n, n_bm, nonzerolist;

#  4. (a) Verify gcd (degress of b_m z^m) = 1
    # if only one power > 1, test 5 will fail anyway, since it
    # will have factors
    if (igcd(op(bm_deg1)) <> 1)
    then
        return false
    end if;

    n := nops(all_coeffs);
    n_bm := nops(bm_coeffs);
    maximum := max(op(bm_deg1));
#  5. (a) For all d>1,
#         m_d = max {m not congruent 0 mod d: b_m <> 0}
#         exists and b_{m_d} > 0
    for d from 2 to maximum do
        comparelist := select(proc(x,d) modp(x, d) <> 0 end,
                              bm_deg1, d);
        if (nops(comparelist) = 0)
        then
            return false
        end if;
        # XXX: should only be one entry in comparelist,
        # but to be on the safe side...
        msubd := max(op(comparelist));
        msubdpos := select(proc(x,uu,max) uu[x]=max end,
                           [$1..n_bm], bm_deg1, msubd);
```

```
        msubdcoeff := seq(bm_coeffs[i], i=msubdpos);
        if (min(op(msubdcoeff)) <= 0)
        then
            return false
        end if
    end do;

#  6. (b) Find E = { k_i+k_j | det (k_i l_i \ k_j l_j) <> 0} and
#         verify it's non-empty
#  7. K = max E
    K := -1;
    for i from 1 to n do
        # only compare with ones that might raise the maximum
        comparelist := select(proc(x,uu,min) uu[x]>min end,
                              [$1..n], deg1, K-deg1[i]);
        for j in comparelist do
            if (deg1[i]*deg2[j]-deg1[j]*deg2[i] <> 0)
            then
                # previous j might have raised K
                if (deg1[i] + deg1[j] > K)
                then
                    K := deg1[i] + deg1[j]
                end if
            end if
        end do
    end do;
    if (K = -1)
    then
        return false
    end if;

#  8. Find I = { (i,j) | k_i+k_j = K }
    isum := 0;
    for i from 1 to n do
        comparelist := select(proc(x,uu,max) uu[x]=max end,
                              [$i+1..n], deg1, K-deg1[i]);
#  9. (b) Verify
#         sum_{(i,j) in I} p_i p_j det(k_i l_i \ k_j l_j)^2 > 0
        for j in comparelist do
            isum := isum + all_coeffs[i] * all_coeffs[j]
                    * (deg1[i]*deg2[j]-deg1[j]*deg2[i])^2
        end do
    end do;
```

```
    if (isum <= 0)
    then
        return false
    end if;
# 10. (c) Verify max { k_j | p_j <> 0 } < 3K/5
    nonzerolist := select(proc(x,uu) uu[x] <> 0 end, [$1..n],
                          all_coeffs);
    maximum := -1;
    for i in nonzerolist do
        if (deg1[i] > maximum)
        then
            maximum := deg1[i]
        end if
    end do;
    if (maximum >= 3*K/5)
    then
        return false
    end if;

    true
end proc;
```

`extadm.maple` contains the main routine, *extadmtest*. *extadmtest* splits an input function into components according to the functions structure and calls itself recursively on the components. In detail:

- Check if input is bivariate.
- Check if input tends to infinity for $var1 \to \infty$ and $var2 = 1$.
- Exclude polynomials and rational polynomials.
- If the input is a product, check each of its factors; if it contains a factor that is neither a polynomial, Hayman-admissible, nor extended admissible, or if none of the factors is extended admissible, report that the input is not extended admissible, otherwise say it is.
- If the input is a power of a constant, return false if the base is less than or equal to 0, equal to 1, or equal to infinity; otherwise, call *extadmtest* on a corresponding exp-function. If the base is not constant, report an error.
- If the input is a function, try to look it up in the function table, which is by default only populated with a test for exp, *ea_exp*, but can be expanded by the user with *ea_addfunc*. Call that function if it exists, or return an error.

- Return true if the input is a sum of polynomials and an extended admissible function. If there is any other kind of summand, return an error. Also return an error if two or more extended admissible functions are added together.
- Report an error, since the structure of this function is not supported.

The *ea_exp* function does the following:

- If the argument of exp is a polynomial, run *ea_polcheck* from `polcheck.maple` on it.
- If the argument is a sum, split it in two summands, one with all the summands that contain *var2*, and the rest. If the summand without *var2* is Hayman-admissible, and the other summand is a polynomial that tends to infinity for $var1 \to \infty$ and $var2 = 1$, the return true, otherwise continue.
- If the argument is a product, split it in three parts, one part for factors only containing *var1*, one for those only containing *var2*, and one for the rest. If the *var1*-factor is not Hayman-admissible, skip the rest of this item. If the other factors are 1, return false. If the other factors are polynomials in *var1* and *var2*, return true. If there is no term with both *var1* and *var2*, and the term with *var2* satisfies the second condition of theorem 5, return true. Otherwise, continue.
- Test if the argument is an extended admissible function.

There are also some smaller helper functions:

- *haymanadm(fct, var)* tests if a function is Hayman-admissible by using an internal function of *equivalent* from the algolib library (see [Sal91]).
- *isanalytic(fct, var)* tests for analyticity by trying to compute the Taylor series expansion around 1.
- *tendstoinfinity(fct, var1, var2)* tries to determine if *fct* tends to infinity for $var1 \to \infty$ and $var2 = 1$ by interpreting the output of MAPLE's *limit* function.

Here's the corresponding MAPLE code:

```
# -*- maple -*-
#
# $Id: extadm.maple,v 1.35 2004/05/13 22:33:20 wiz Exp $
#

extadm := module()
```

```
description "extended admissibility tests";
local check_condition, ea_functab, ea_exp, ea_polcheck,
    haymanadm, isanalytic, tendstoinfinity;
export extadmtest, ea_addfunc;
option package;

ea_addfunc := proc(fname::symbol, impl)
    ea_functab[fname] := impl
end proc;

# Input: Function in var1 and var2
# Output: true if extended admissible, false if not
#
# Algorithm:
# Separate into smaller pieces according to structure, and
# call recursively; use separate deciding functions like
# ea_polcheck in some cases.
extadmtest := proc(fct::algebraic, var1::name, var2::name)
    local found_extadm, subfct, e, infcheck;

    if (not has(fct, var1) or not has(fct, var2)) then
        userinfo(3, extadm, "function is univariate or "
                 "constant", fct);
        return false
    end if;

    infcheck := tendstoinfinity(fct, var1, var2);
    if (infcheck = false) then
        # for central limit theorem, fct(var1, 1) must tend
        # to infinity
        userinfo(3, extadm, "does not tend to +infinity: ",
                 fct);
        false
    elif type(fct, polynom(anything, {var1, var2})) then
        # no chance for a central limit theorem if there's a
        # limited number of coefficients
        userinfo(3, extadm, "polynomial");
        false
    elif type(fct, ratpoly(anything, {var1, var2})) then
        userinfo(3, extadm, "rational polynomial");
        false
    elif type(fct, '*') then
        found_extadm := false;
```

```
        for subfct in fct do
            if type(subfct, polynom(anything, {var1, var2}))
            then
                ;
            elif extadmtest(subfct, var1, var2) then
                found_extadm := true
            elif haymanadm(subfct, var1) then
                ;
            else
                # product contains a factor that is neither
                # extended admissible, nor Hayman admissible,
                # nor a polynomial -> cannot decide
                error "product contains factor that is "
                      "neither ext-adm, hayman-adm, nor "
                      "polynomial"
            end if;
        end do;
        if not found_extadm then
            userinfo(3, extadm, "product contains only "
                     "hayman-admissible factors and "
                     "polynomials");
            false
        else
            true
        end if
    elif type(fct, `^`) then
        # realconstant ^ function
        if (type(op(1, fct), realcons)) then
            if (op(1, fct) <= 0 or op(1, fct) = 1
                or op(1, fct) = infinity) then
                userinfo(3, extadm, "base less than 0, or "
                         "equal to 0, 1, or infinity",
                         op(1, fct));
                false
            elif (op(1, fct) < 1) then
                extadmtest(exp(-op(2, fct)), var1, var2)
            else
                extadmtest(exp(op(2, fct)), var1, var2)
            end if
        else
            error "Not implemented: extadm of "
                  "not-real-constant^anything", fct
        end if
```

```
elif type(fct, 'function') then
    e := op(0, fct);
    if assigned (ea_functab[e]) then
        try
            ea_functab[e](fct, var1, var2)
        catch:
            error "%1(%2, %3, %4) failed: %5",
                  ea_functab[e],
                  fct, var1, var2, lasterror;
        end try
    else
        error "Not implemented: extadm of %1", e
    end if
elif type(fct, `+`) then
    found_extadm := false;
    for subfct in fct do
        if type(subfct, polynom(anything, {var1, var2}))
        then
            ;
        elif extadmtest(subfct, var1, var2) then
            if not found_extadm then
                found_extadm := true
            else
                error "sum contains two or more ext-adm "
                      "functions", fct
            end if
        else
            # sum contains a factor that is neither
            # extended admissible nor a polynomial
            # -> cannot decide
            error "sum contains summand that is neither "
                  "ext-adm nor polynomial", fct
        end if;
    end do;
    if found_extadm then
        userinfo(3, extadm,
                 "sum of polynomial and ext-adm")
    else
        # XXX: shouldn't happen
        userinfo(3, extadm, "polynomial")
    end if;
    found_extadm
else
```

```
        error "Invalid expression %1", fct
    end if
end proc:


ea_exp := proc(fct, var1, var2)
    local arg, fac, summand, var1f, var2f, varsf;

    ASSERT(op(0, fct) = exp, "invalid call");

    arg := op(1, fct);
    if type(arg, polynom(anything, {var1, var2})) then
        # check if polynomial conditions are fulfilled
        userinfo(3, extadm, "testing polynomial in exponent");
        return ea_polcheck(arg, var1, var2)
    elif type(arg, '+') then
        # split in parts belonging to var1, var2, and both
        var1f := 0;
        varsf := 0;
        for summand in arg do
            if not has(summand, var2) then
                var1f := var1f + summand
            else
                varsf := varsf + summand
            end if
        end do;
        if (var1f <> 0 and haymanadm(var1f, var1)
            and varsf <> 0
            and type(varsf, polynom(anything, {var1, var2}))
            and limit(subs(var2=1,
                      diff(varsf, var2)+diff(varsf, var2$2)),
                      var1=infinity) = infinity) then
                userinfo(3, extadm, "exp(hayman(var1)+"
                         "positivepolynomial(var1, var2))");
                return true
        end if
    end if;

    if type(arg, '*') then
        # split in parts belonging to var1, var2, and both
        var1f := 1;
        var2f := 1;
        varsf := 1;
```

```
        for fac in arg do
            if has(fac, var1) and not has(fac, var2) then
                var1f := var1f * fac
            elif not has(fac, var1) and has(fac, var2) then
                var2f := var2f * fac
            else
                varsf := varsf * fac
            end if
        end do;
        if (var1f <> 1 and haymanadm(var1f, var1)) then
            if (var2f*varsf = 1) then
                # only one variable, really
                userinfo(3, extadm, "exp(hayman(var1))");
                return false
            elif type(var2f*varsf, polynom(anything,
                                           {var1, var2})) then
                userinfo(3, extadm,
                         "exp(hayman(var1)*poly(var1, var2))");
                return true
            elif (varsf = 1 and isanalytic(var2f, var2)
                  and subs(var2=1, var2f) > 0
                and subs(var2=1, diff(var2f, var2)
                         +diff(diff(var2f, var2), var2)
                     - diff(var2f, var2)^2/var2f) > 0) then
                userinfo(3, extadm,
                         "exp(hayman(var1)*g(var2)) "
                         "with g(var2) analytic at 1 and "
                         "additional properties");
                return true
            end if
        end if
    end if;

    extadmtest(arg, var1, var2)
end proc:

ea_addfunc('exp', ea_exp);

haymanadm := proc(fct, var)
    member(`equivalent/saddlepoint/H_HS`(fct, var),{'H','HS'})
end proc:

# Input: Function in var
```

```
    # Output: true if analytic near 1
    #
    # Algorithm: expand into series around 1 and check if it's a
    # Taylor series

    isanalytic:=proc(fct, var)
    try
        type(series(fct,var=1),taylor)
    catch:
        false
    end try
    end:

    # Input: Function in var1 and var2
    # Output: true if limit of var2=1,
    #         var1->infinity is +infinity;
    #         false if not, or nothing if undecided.
    # Algorithm: run limit and parse its output

    tendstoinfinity:=proc(fct, var1, var2)
    local res;
    try
        res:=limit(subs(var2=1, fct), var1=infinity);
        if (type(res, realcons) and res <> infinity) then
            return false
        elif (res = infinity) then
            return true
        end if
    catch :
    end try
    end:

$include "polcheck.maple";

end module:

savelib('extadm');
quit
```

Bibliography

[ASU86] Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools.* Addison-Wesley Longman Publishing Co., Inc., 1986.

[BR83a] Edward A. Bender and L. Bruce Richmond. Central and local limit theorems applied to asymptotic enumeration. II. Multivariate generating functions. *J. Combin. Theory Ser. A*, 34(3):255–265, 1983.

[BR83b] Edward A. Bender and L. Bruce Richmond. Central and local limit theorems applied to asymptotic enumeration. II. Multivariate generating functions. *J. Combin. Theory Ser. A*, 34(3):255–265, 1983.

[BR86] Edward A. Bender and L. Bruce Richmond. A generalisation of Canfield's formula. *J. Combin. Theory Ser. A*, 41(1):50–60, 1986.

[BR96] Edward A. Bender and L. Bruce Richmond. Admissible functions and asymptotics for labelled structures by number of components. *Electron. J. Combin.*, 3(1):Research Paper 34, approx. 23 pp. (electronic), 1996.

[dB81] N. G. de Bruijn. *Asymptotic methods in analysis.* Dover Publications Inc., New York, third edition, 1981.

[DG99] Michael Drmota and Bernhard Gittenberger. The distribution of nodes of given degree in random trees. *J. Graph Theory*, 31(3):227–253, 1999.

[Drm94a] Michael Drmota. Asymptotic distributions and a multivariate Darboux method in enumeration problems. *J. Combin. Theory Ser. A*, 67(2):169–184, 1994.

[Drm94b] Michael Drmota. A bivariate asymptotic expansion of coefficients of powers of generating functions. *European J. Combin.*, 15(2):139–152, 1994.

[Drm97] Michael Drmota. Systems of functional equations. *Random Structures Algorithms*, 10(1-2):103–124, 1997. Average-case analysis of algorithms (Dagstuhl, 1995).

[DZ89] Nachum Dershowitz and Shmuel Zaks. Patterns in trees. *Discrete Appl. Math.*, 25(3):241–255, 1989.

[FGM97] Philippe Flajolet, Xavier Gourdon, and Conrado Martínez. Patterns in random binary search trees. *Random Structures Algorithms*, 11(3):223–244, 1997.

[FS80a] Philippe Flajolet and Jean-Marc Steyaert. On the analysis of tree-matching algorithms. In *Automata, languages and programming (Proc. Seventh Internat. Colloq., Noordwijkerhout, 1980)*, volume 85 of *Lecture Notes in Comput. Sci.*, pages 208–219. Springer, Berlin, 1980.

[FS80b] Philippe Flajolet and Jean-Marc Steyaert. On the analysis of tree-matching algorithms. In *Trees in algebra and programming (Proc. 5th Lille Colloq., Lille, 1980)*, pages 22–40. Univ. Lille I, Lille, 1980.

[FS90] Philippe Flajolet and Michèle Soria. Gaussian limiting distributions for the number of components in combinatorial structures. *J. Combin. Theory Ser. A*, 53(2):165–182, 1990.

[FS93] Philippe Flajolet and Michèle Soria. General combinatorial schemas: Gaussian limit distributions and exponential tails. *Discrete Math.*, 114(1-3):159–180, 1993. Combinatorics and algorithms (Jerusalem, 1988).

[FS02] Philippe Flajolet and Robert Sedgewick. Analytic combinatorics – symbolic combinatorics. *Manuscript*, pages 186+viii, May 2002. Available as `http://algo.inria.fr/flajolet/Publications/FlSe02.ps.gz`.

[Gar95] Danièle Gardy. Some results on the asymptotic behaviour of coefficients of large powers of functions. *Discrete Math.*, 139(1-3):189–217, 1995. Formal power series and algebraic combinatorics (Montreal, PQ, 1992).

[GJ83] I. P. Goulden and D. M. Jackson. *Combinatorial enumeration.* A Wiley-Interscience Publication. John Wiley & Sons Inc., New York, 1983. With a foreword by Gian-Carlo Rota, Wiley-Interscience Series in Discrete Mathematics.

[GM04] Bernhard Gittenberger and Johannes Mandlburger. Hayman-admissible functions in several variables. *Manuscript*, 2004.

[GR92] Zhicheng Gao and L. Bruce Richmond. Central and local limit theorems applied to asymptotic enumeration. IV. Multivariate generating functions. *J. Comput. Appl. Math.*, 41(1-2):177–186, 1992. Asymptotic methods in analysis and combinatorics.

[Har67] L. H. Harper. Stirling behavior is asymptotically normal. *Ann. Math. Statist.*, 38:410–414, 1967.

[Hay56] W. K. Hayman. A generalisation of Stirling's formula. *J. Reine Angew. Math.*, 196:67–95, 1956.

[HP73] Frank Harary and Edgar M. Palmer. *Graphical enumeration.* Academic Press, New York, 1973.

[HS68] Bernard Harris and Lowell Schoenfeld. Asymptotic expansions for the coefficients of analytic functions. *Illinois J. Math.*, 12:264–277, 1968.

[Hwa96] Hsien-Kuei Hwang. Large deviations for combinatorial distributions. I. Central limit theorems. *Ann. Appl. Probab.*, 6(1):297–319, 1996.

[Hwa98] Hsien-Kuei Hwang. On convergence rates in the central limit theorems for combinatorial structures. *European J. Combin.*, 19(3):329–343, 1998.

[Lal02] S. Lalley. Random walks on infinite free products and infinite algebraic systems of generating functions. *Manuscript*, August 2002. Available as `http://www.stat.uchicago.edu/~lalley/Papers/RWFP.pdf`.

[Lon04] Stefano Lonardi. Pattern matching pointers. *Internet*, 2004. `http://www.cs.ucr.edu/~stelo/pattern.html`.

[MM78] A. Meir and J. W. Moon. On the altitude of nodes in random trees. *Canad. J. Math.*, 30(5):997–1015, 1978.

[Ott48] Richard Otter. The number of trees. *Ann. of Math. (2)*, 49:583–599, 1948.

[RS75] R. W. Robinson and A. J. Schwenk. The distribution of degrees in a large random tree. *Discr. Math.*, 12:359–372, 1975.

[RSSVdH96] Dan Richardson, Bruno Salvy, John Shackell, and Joris Van der Hoeven. Asymptotic expansions of exp-log functions. In Y. N. Lakshman, editor, *ISSAC'96*, pages 309–313. ACM Press, 1996. Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation. July 24–26, 1996. Zürich, Switzerland.

[Ruc88] Andrzej Ruciński. When are small subgraphs of a random graph normally distributed? *Probab. Theory Related Fields*, 78(1):1–10, 1988.

[Sal91] B. Salvy. Examples of automatic asymptotic expansions. *SIGSAM Bulletin*, 25(2):4–17, April 1991.

[SF83] Jean-Marc Steyaert and Philippe Flajolet. Patterns and pattern-matching in trees: an analysis. *Inform. and Control*, 58(1-3):19–58, 1983.

[Sha93] John Shackell. Inverses of Hardy L-functions. *Bull. London Math. Soc.*, 25(2):150–156, 1993.

[SS99] Bruno Salvy and John Shackell. Symbolic asymptotics: multiseries of inverse functions. *J. Symbolic Comput.*, 27(6):543–563, 1999.

Acknowledgments

First of all, I would like to thank my thesis supervisor Michael Drmota for supporting me to work on mathematics, for opening the opportunity to stay at INRIA, for providing useful advice, and for supervising my thesis. I would also like to thank Bernhard Gittenberger for many interesting conversations starting from simple questions and usually ending in complex subjects.

I am thankful for being invited to stay at the Algorithms Project at INRIA Rocquencourt, where Frédéric Chyzak, Virginie Collette, Philippe Flajolet, Bruno Salvy, and other members of the Algorithms Project made me feel very welcome and both explicitly and implicitly taught me quite a lot.

I am grateful to the Austrian Science Foundation FWF for giving financial support during my thesis (via grants S8302-MAT and P16053-N05).

Very special thanks go to my family (including Merlin) and my friend Ute for their constant support.

Lebenslauf

Ich wurde am 18. August 1975 als Sohn von Ilse Klausner, geb. Moser, und Herbert Klausner in Wien geboren. Von 1981 bis 1985 besuchte ich die Volksschule Mauerbach, von 1985 bis 1993 dann das Schottengymnasium, Wien 1, an dem ich im Juni 1993 mit Auszeichnung maturierte.

Im September 1993 immatrikulierte ich an der Technischen Universität Wien das Studium Technischen Mathematik. Ich schloss das Diplomstudium Technische Mathematik im Juni 2000 mit ausgezeichnetem Erfolg ab.

In den Jahren 1994 bis 2003 war ich am Institut für Angewandte und Numerische Mathematik der Technischen Universität Wien als Studienassistent tätig. Von Oktober 2000 bis November 2002 war ich Forschungsassistent im FWF-Schwerpunktsprojekt "Statistical Properties of Number Systems" unter der Leitung von Prof. Michael Drmota. Seit Dezember 2002 bin ich im FWF-Forschungsprojekt "Automatische Entwicklung erzeugender Funktionen" von Prof. Bernhard Gittenberger angestellt, unterbrochen von meinem Zivildienst, den ich von Februar 2003 bis Jänner 2004 am Bundesministerium für Inneres ableistete.

Wien, Mai 2004 Thomas Klausner