Theoretical Computer Science 79 (1991) 37-109
North-Holland

Automatic average-case analysis of algorithms

Philippe Flajolet
INRIA Rocquencourt, F-78153 Le Chesnay, France

Bruno Salvy
INRIA and LIX, Ecole Polytechnique, F-91128 Palaiseau, France

Paul Zimmermann
INRIA Rocquencourt, F-78153 Le Chesnay, France

Abstract

Flajolet, P., Salvy, B. and Zimmermann, P., Automatic average-case analysis of algorithms, Theoretical Computer Science 79 (1991) 37-109.

Many probabilistic properties of elementary discrete combinatorial structures of interest for the average-case analysis of algorithms prove to be decidable. This paper presents a general framework in which such decision procedures can be developed. It is based on a combination of generating function techniques for counting, and complex analysis techniques for asymptotic estimations.

We expose here the theory of exact analysis in terms of generating functions for four different domains: the iterative/recursive and unlabelled/labelled data type domains. We then present some major components of the associated asymptotic theory and exhibit a class of naturally arising functions that can be automatically analyzed.

A fair fragment of this theory is also incorporated into a system called Lambda-Upsilon-Omega. In this way, using computer algebra, one can produce automatically non-trivial average-case analyses of algorithms operating over a variety of "decomposable" combinatorial structures.

At a fundamental level, this paper is part of a global attempt at understanding why so many elementary combinatorial problems tend to have elementary asymptotic solutions. In several cases, it proves possible to relate entire classes of elementary combinatorial problems whose structure is well defined with classes of elementary "special" functions and classes of asymptotic forms relative to counting, probabilities, or average-case complexity.

1. Introduction

This paper presents a systematic framework in which combinatorial enumerations and probabilistic properties of combinatorial structures can be studied formally.

The analysis system that we propose is "algebraically complete" with respect to a large category of so-called decomposable data types and their associated

algorithms. It is also "asymptotically complete" with respect to several subclasses of decomposable problems. Thus a number of statistical and computational properties of combinatorial structures can be systematically decided, even in their precise asymptotic form.

A correlate of this is the possibility of designing an automatic analyzer of average case performance for several interesting classes of algorithms and programmes. Based on this theory, we have actually built a prototype system called *Lambda-Upsilon-Omega* ($\Lambda\Upsilon\Omega$) that is capable of producing rather non-trivial average-case analyses of algorithms.

Here is a small sample of properties amenable to automatic analysis in this context: (i) the average number of cycles in a random permutation of n elements is $\sim\log n$ and the probability that such a permutation has no 1-cycle is $\sim e^{-1}$; (ii) path length in a random heap-ordered tree of n elements is on average $\sim 2n \log n$, which represents also the comparison cost of Quicksort; (iii) path length in a random (uniform) plane tree is $\sim \frac{1}{2}n\sqrt{\pi n}$; (iv) the symbolic differentiation algorithms of computer algebra gain on average a factor of $O(\sqrt{n})$ if shared representations (i.e., dags) are used, etc.

The paper consists of two major parts that reflect the two components of the theory. The first one, the "algebraic" component, deals with exact counting through the algebra of generating functions. The second one, the "analytic" component, uses analytic properties of these generating functions in order to recover relevant asymptotic informations.

1.1. Algebraic enumeration

For the class of decomposable combinatorial structures under consideration, it is possible to compile automatically structural specifications into equations over *counting generating functions*. These equations represent in a compact format either explicit or else recursive forms of count sequences. For the associated algorithms, we introduce generating functions of average costs called *complexity descriptors*, and we provide similar translation mechanisms from programme specifications to these complexity descriptors.

The equations that one generates in this way are meaningful in the sense that all coefficients of generating functions—providing either the number of combinatorial structures of size n or the average case complexity of algorithms over random data of size n—are computable in time that is polynomial in n.

1.2. Asymptotic analysis

It is known from classical analysis and analytic number theory that the asymptotic growth of coefficients of a series is determined by analytic properties of the series (viewed then as an analytic function of a complex argument). In this domain, singularities and saddle points play an essential role.

One of the major benefits of the generating function approach is to associate well identified classes of special functions to well characterized classes of combinatorial structures and programmes. We can then systematically relate classes defined by special combinatorial constructions, classes of special functions with specific analytic properties, and asymptotic properties of structures.

We first illustrate the principles of our approach by discussing two examples drawn from the classical theory of formal languages and enumerations.

Example 1.1 (*Regular events and finite automata*). Combinatorial structures defined by regular languages and finite automata have rational generating functions [9, 27, 71]. The counting sequences accordingly satisfy linear recurrences with constant coefficients. From elementary analysis, we know that a rational generating function $f(z)$ admits a partial fraction decomposition, so that its coefficients f_n have an explicit form as "exponential polynomials",

$$f_n = \sum_k P_k(n)\omega_k^n, \tag{1}$$

for a finite family of polynomials $P_k(x)$ and a family of algebraic numbers ω_k.

In other words, for this restricted class of devices, we are able to predict in which class of formulae, either exact or asymptotic, counting sequences and expected values of parameters are going to fall. For instance, a priori, the problem of run length statistics—What is the probability that a random binary string of length n contains no run of k consecutive 1s?—lies in this class for each fixed k. (See [28, XIII.7] for a classical introduction.) Further analytic properties are available; most notably the Perron-Frobenius theory [11, 51] predicts that the ωs of largest modulus in (1) have arguments that are commensurable with π, a fact that further restricts the range of fluctuations (due to complex ωs) to those that are asymptotically periodic. The whole theory [27] is a combinatorial analogue of the classical theory of Markov chains.

Example 1.2 (*Context-free languages*). Trees and various types of lattice paths can be described by context-free grammars [9, 10, 26, 56, 71]. The corresponding generating functions are algebraic, as follows by the classical Chomsky-Schützenberger theorem [16]. Accordingly, the counting sequences satisfy linear recurrences with polynomial coefficients ("P-recursive" sequences). An algebraic function $f(z)$ has only algebraic singularities; from this fact, its coefficients f_n are found to be asymptotic to a sum of "algebraic" elements,

$$f_n \sim \sum_j c_j n^{r_j/s_j} \omega_j^n, \quad \text{with } r_j, s_j \in \mathbb{N} \text{ and } c_j, \omega_j \in \mathbb{C}. \tag{2}$$

This again characterizes the allowed types of probabilistic behaviours for all combinatorial processes that can be described by context-free languages. This asymptotic

theory of context-free languages was worked out in [30]; it constitutes a combinatorial analogue of the probabilistic theory of branching processes.

The examples above illustrate a typical situation: A class of combinatorial processes (finite automata, context-free grammars) is associated to a class of special functions (rational functions, algebraic functions). Analytic properties of these functions, especially the nature of their singularities, are well characterized (poles, algebraic singularities). This in turn entails that the major asymptotic properties of the original processes are fully characterized (exponential polynomials, algebraic elements), and thus decidable and computable.

Our objective here is only to extend this philosophy to an appreciably larger class of combinatorial structures. Our approach to the automatic analysis of algorithms and data structures is thus a "pipe" between two technologies: the symbolic (or formal) methods of combinatorial enumerations and the complex-analytic methods of asymptotic analysis.

1.3. The automatic analyzer, Lambda-Upsilon-Omega

An automatic analyzer, the $\Lambda\Upsilon\Omega$ system[1] implements the fundamental theoretical ideas that we have just outlined. One of the primary aims of the $\Lambda\Upsilon\Omega$ system is to provide a tool for aiding the analysis of various types of algorithms. It is also meant to experiment with the descriptive power of the theories developed here.

The system itself is described elsewhere by its two implementers (see also Fig. 1): Zimmermann is responsible for the algebraic analyzer [89] and Salvy has designed the asymptotic analyzer [72]. The algebraic analyzer compiles data type and procedure specifications into equations over generating functions (that are counting generating functions or complexity descriptors). It is implemented in CAML. The analytic analyzer is an extensive collection of routines that manipulate generalized asymptotic series and perform asymptotic analysis of generating functions, producing final asymptotic results. The interface between the two components is ensured by the "solver" module that relies on computer algebra capabilities for solving elementary equations (linear equations; algebraic equations; simple differential equations).

As of 1990, the programme has over 10 000 instructions, partly in a high level functional language—CAML, a dialect of ML [84]—and partly in a computer algebra language—Maple [15].

In the present paper, we have used the $\Lambda\Upsilon\Omega$ system in order to produce what is called *automatic theorems*. In principle, an automatic theorem is a statement that is derived automatically from formal specifications by the logical framework exposed in this paper. We have however decided to enforce a stricter discipline and retain,

[1] The name Lambda-Upsilon-Omega derives from the Greek verb root λύω which means "I solve" and which is at the heart of the word *ana*-ly-*sis*. $\Lambda\Upsilon\Omega$ should be pronounced as lüo, and its Latin rendering is LUO.

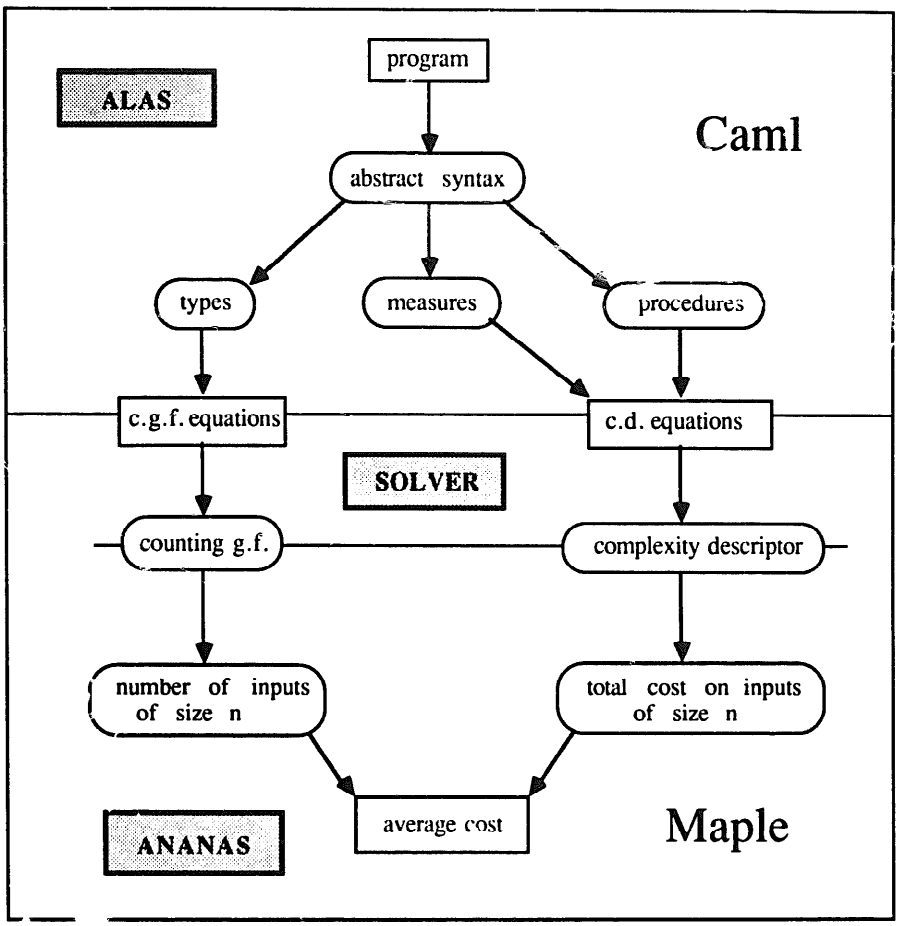

Fig. 1. The general structure of the $\Lambda\gamma^{\Omega}$ system. The diagram shows the three major components of the system, the Algebraic Analyzer (ALAS), the Solver, and the Analytic Analyzer (ANANAS).

for the twenty odd automatic theorems given here, only those results that are *also* derived automatically by the $\Lambda\gamma^{\Omega}$ programme. This attitude is meant to emphasize that *the effective procedures described here are also practically implementable.* The human interaction is limited to trivial editing of resulting formulae, and the reader can consult the $\Lambda\gamma^{\Omega}$ Cookbook [34] for complete listings of analyses that were produced by the system in 1989.

The current functionality of the $\Lambda\gamma^{\Omega}$ system is as follows:

- for exact counting in terms of GFs, the rules of Section 3 relating to labelled/unlabelled recursive/iterative structures and their associated algorithms;
- for the solver, the amount of algebraic manipulation discussed in Section 5.2 regarding substitution, linear and quadratic equations;
- for the analytic analyzer, the theory of algebraic-logarithmic (AL) structures of Section 4, plus the extensions relating to saddle point methods described in Section 5.3.

This specification of a precise mathematical level of expertise also ensures that our "automatic theorems" actually represent automatic results (and not a haphazard collection of ad hoc recipes put into a large programme!).

The capabilities of the logical framework—and its corresponding implementation—extend to a number of combinatorial and computational problems. Examples are spread over this paper and in the $\Lambda\Upsilon\Omega$ Cookbook, and they can be organized roughly into three categories.

(i) *Regular languages and finite automata.* Addition chains and related optimization problems; long term behaviour of finite state controls in systems; various combinatorial problems (compositions, partitions, runs in sequences) that are expressible in terms of regular languages.

(ii) *Terms and trees*: Differentiation algorithms, higher derivatives; some simplification algorithms; a class of term rewriting systems; partial analyses of unification algorithms and related pattern occurrence problems.

(iii) *Combinatorial problems*: Random tree problems; mappings and functional graphs; partitions and ordered partitions; Banach's matchbox problem; some special permutation problems.

For instance, the problems on addition chains are of some relevance to integer primality testing using elliptic curves and a small percentage of computer time was gained using optimizations guided by the $\Lambda\Upsilon\Omega$ system [60]. In the next section, we show that the symbolic computation of derivatives has an average cost over (random) expressions of size n that is $O(n^{3/2})$. The $\Lambda\Upsilon\Omega$ system was first used to verify the conjecture that the cost of a dth order derivative varies on average like $O(n^{1+d/2})$, for $d = 2, 3$.

Within the system's capabilities, we find many examples of rewriting systems that belong to the class of so-called regular systems [17, 75]. $\Lambda\Upsilon\Omega$ was used to check several of the corresponding analyses of Michèle Soria's thesis in this context [75], or in the context of random functional graph problems (model sensitivity issues of [31, 75]).

Several automatic analyses on pattern occurrence problems in random trees have been used in the performance study of unification algorithms given in [2]. One of the authors' little rewards occurred when they discovered that the $\Lambda\Upsilon\Omega$ system was capable of "doing" a paper published as a note (on injective partial transformations) in the journal *Discrete Mathematics* [12], and even obtain more complete results including asymptotics.

A few historical comments on the ancestry of these ideas are now in order.

Formal or symbolic methods in combinatorial enumerations take their roots in actual enumeration practice in various domains. However, the first general theories seem to have started in the 1960s. The Chomsky-Schützenberger theory of context-free languages [16] is amongst the first traceable sources where very systematic correspondences are exploited between combinatorial structures (words and languages) and generating functions. Other sources are the theory of graphical enumerations [41], Rota's theory of generating functions [69, 70], Bender and Goldman's theory of "prefabs" [5], or Foata's theory of the partitional complex [38]. Each of these theories deals with combinatorial structures that are either labelled or unlabelled, but not both. A unifying framework comprising both types

was proposed by Joyal [49] in 1981. Finally, a systematic exposition of combinatorial enumerations in this context is the subject of a book by Jackson and Goulden [39]. Other relevant references are Comtet's book [24] and Stanley's works [76, 78].

On the analytical side, the tradition of relating analytic properties of a function to asymptotic properties of its Taylor coefficients is older. Its roots lie in part in classical analysis (e.g., Darboux's method), and in part in analytic number theory (e.g., the additive theory of partitions). We shall simply refer to classical treatises like those of De Bruijn and Olver [25, 63] for a general treatment. The two major techniques that we use are: (i) singularity analysis for functions that do not grow too fast; (ii) saddle point techniques for functions with a more violent growth at either a finite or infinite distance.

Turning from theory to practice, the first automatic performance analyzer that we are aware of was built by Wegbreit in the early 1970s [83]. Wegbreit's pioneer system, Metric, aimed at deriving closed form expressions for execution behaviour of programs, and it included modules to carry out average-case analysis. However, the underlying principles were Markovian approximations (fixing the probabilities of tests to constants) and an amount of symbolic manipulation limited to linear recurrences with constant coefficients.

The present work is based on works of Flajolet and Steyaert [29, 36, 79, 37]. In particular, the articles [36, 37] proposed a complexity calculus for a class of simple recursive programmes over tree structures; that calculus in turn gave rise to a prototype implementation which is described in [33] and which constitutes a direct ancestor of the current $\Lambda\Upsilon\Omega$ system.

The algebraic part of our system also bears some resemblance to the interesting theory of labelled grammars due to Greene [40]. Greene's theory concerns primarily combinatorial enumerations; he used it for constructing an automatic generator of random combinatorial structures, but he did not pursue the formal side of the analysis of algorithms. The influence of Greene's excellent work is to be found not so much in the core of our system, but rather in several extensions (automatic generation of random structures, counting of labelled structures with order constraints). The Darwin system developed by Bergeron and Cartier [6, 7] determines generating functions within Joyal's enumerative theory; it is a distant cousin to the algebraic engine of the $\Lambda\Upsilon\Omega$ system. A system also designed to enumerate combinatorial structures that is based on the Jackson-Goulden formalism is reported in [58].

Amongst other proposals, we mention the approach of Hickey and Cohen to the automatic analysis of programmes in [18, 47]; it relies in part on Ramshaw's frequency system (which corresponds roughly to Floyd-Hoare complexity assertions [66]) and in part on Kozen's semantics of probabilistic programmes. The Hickey-Cohen framework appears to have a rich expressive power, however it is not clear how a complete calculus (with simplification rules and normal forms) can be developed. Perhaps most of the interest of their approach lies in correctness verification of complexity assertions. Zimmermann in [92] developed a system,

Complexa, based on recurrence relations which extends Wegbreit's approach since more elaborate control mechanisms are allowed and more complicated recurrences are dealt with by the algebraic unit.

Finally, a few other works have dealt with automatic worst-case analysis of algorithms, a rather different domain. For this, we refer to the work of Le Métayer [55] and references therein.

2. An example: symbolic differentiation

In order to illustrate the range of problems that we want to attack, we start by presenting an example of a simplified programme for computing derivatives of formal expressions.

Such algorithms form the core of classical algebra systems and provide classical programming exercises [52, p. 336–340]. Our programme is a simple tree rewriting process that recursively implements the differentiation rules

$$\begin{aligned} Dx &\Rightarrow 1 \\ De^f &\Rightarrow e^f \times (Df) \\ D(f+g) &\Rightarrow (Df)+(Dg). \end{aligned}$$

It should be stressed that the analysis that follows was produced automatically by the $\Lambda\Upsilon^{\Omega}$ system. (We follow the standard conventions of the system for specifying data types and algorithms, but these should be transparent in the various examples.)

We consider a programme that operates on formal expressions composed simply of exponentials (expo), sums (plus), and a simple variable (x).

```
type Expression=expo Expression|plus Expression Expression|x;
  expo, plus, x=atom(1);
```

The data type specification is recursive. Its presentation resembles a classical context-free grammar, and `atom(1)` refers to objects that are atomic (i.e., terminals in standard context-free language parlance) with size equal to 1 (the standard size for an atomic object). A symbolic expression like $x+e^{e^x+x}$ is thus of the Expression type, with size 7, being represented as

```
plus x expo plus expo x x
```

The differentiation algorithm has a simple top down recursive structure which, in our formalism, we specify as follows:

```
function diff(e : Expression);
begin
  case e of
    (expo,f) :  times(expo(copy(f)),diff(f));
    (plus,f,g) :  plus(diff(f),diff(g));
    x :  one;
  end;
end;
```

The output expressions then belong to a richer set with, in addition, products

(times) and the constant 1 (one) being allowed. The copy procedure creates a carbon copy of its argument:

```
function copy(e : Expression);
begin
  case e of
    (expo,f) :      expo(copy(f));
    (plus,f,g) :    plus(copy(f),copy(g));
    x :             x;
  end;
end;
```

We thus specify a version of the algorithm that computes diff(e) by creating its own independent linked structure.

The algorithm has worst-case complexity $O(n^2)$ when applied to an input expression of size n. This worst case is attained with chains of exponentials. For instance the verbatim output of the differentiation command applied to the sixfold iteration of the exponential function in Maple is:

$$e^x e^{e^x} e^{e^{e^x}} e^{e^{e^{e^x}}} e^{e^{e^{e^{e^x}}}} e^{e^{e^{e^{e^{e^x}}}}}$$

The best-case complexity is clearly $O(n)$.

Our interest in this paper lies in *average-case analysis* of algorithms under *uniform combinatorial models* where all input structures of a given size n are taken equally likely. The differentiation algorithm under consideration operates in a purely recursive fashion over a recursively defined data type and thus it falls under the class of algorithms that our system can approach. For instance, if we specify a complexity measure of 1 for each of the output symbols,

```
measure expo, plus, x, times, one : 1;
```

the cost of the algorithm coincides with the size of the expression that it outputs. In that case, the $\Lambda\Upsilon\Omega$ system automatically produces the answer:

```
Average cost for diff on random inputs of size n is:
```

$$\left(1/3\ \frac{\mathrm{Pi}^{1/2}\ n^{3/2}}{(4/3)^{1/2}}\right) + (5/6\ n) + (O(n^{1/2}))$$

```
Floating point evaluation:
```

$$(.5116633543\ n^{3/2}) + (.8333333333\ n) + (O(n^{1/2}))$$

Thus, we obtain the automatically produced theorem:

Automatic Theorem 1. *The complexity of the differentiation algorithm applied to random expressions of size n is on average*[2]

$$\sqrt{\frac{\pi}{12}}\, n^{3/2} + \frac{5}{6}\, n + O(n^{1/2}). \tag{3}$$

[2] This "automatic theorem" was produced in 75 s of computing time on a machine (Sun3) that performs about 3×10^6 elementary instructions per seconds and has 12×10^6 bytes of core memory.

(Also a full asymptotic expansion in descending powers of $\sqrt{n}$ can be obtained.)

Proof. Let us examine now the way that such an analysis is produced. Let $\overline{\tau\text{diff}}_n$ represent the expected complexity of procedure `diff` over a random expression of size n. The basic equation is

$$\overline{\tau\text{diff}}_n = \frac{\tau\text{diff}_n}{\text{Expression}_n} \tag{4}$$

where Expression_n is the number of expressions of size n while τdiff_n represents the sum of the execution costs of the procedure over all expressions of size n. The analysis problem thus reduces to a counting problem, namely determining Expression_n, and a modified counting problem where we need to find the total cost of the diff-algorithm.

The two problems are attacked by generating function (GF) techniques; see Section 3 for basic definitions. We thus define the GFs.

$$\text{Expression}(z) = \sum_{n\geq 0} \text{Expression}_n \cdot z^n \quad \text{and} \quad \tau\text{diff}(z) = \sum_{n\geq 0} \tau\text{diff}_n \cdot z^n.$$

The first one is the standard *counting generating function* of the expression structures. The second one is called the *complexity descriptor* of the algorithm.

Algebraic enumeration: Translation mechanisms that we are going to review with some detail in the next section imply that these GFs satisfy equations that are direct "images" of the data type and of the algorithm specifications.

First, the recursive nature of the expression type leads to a fixed point equation for $\text{Expression}(z)$

$$\text{Expression}(z) = z(\text{Expression}(z))^2 + z\text{Expression}(z) + z.$$

In this particular case, we found a quadratic equation whose explicit solution is

$$\text{Expression}(z) = \frac{1 - z - \sqrt{1 - 2z - 3z^2}}{2z}.$$

See [24, p. 56] for the classical solution to similar problems.

Second, the complexity descriptor $\tau\text{diff}(z)$ satisfies a simpler equation which rationally relates it to $\text{Expression}(z)$ and $\tau\text{copy}(z)$, and this is again a direct reflection of the recursive specification of the `diff` routine. Finally, we find that all intervening GFs are rational functions in z and in the *singular part*

$$\Delta(z) = \sqrt{1 - 2z - 3z^2}. \tag{5}$$

Asymptotic analysis: The problem is now to extract the coefficients of the GFs. The major idea is to avoid the computation of explicit forms of the coefficients Expression_n, τdiff_n and aim at *direct asymptotic analysis* from the GFs themselves.

It turns out that the *dominant singularities*—the singularities of smallest modulus—of a generating function determine the asymptotic growth of its coefficients. The modulus ρ of the dominant singularity (-ies) contributes for the coefficients a driving

exponential factor of ρ^{-n} while the nature of the singularity is reflected by a subexponential factor. These questions are discussed in Sectior 4.

Here, the singularities of $\Delta(z)$ and hence of Expression(z) are the branch points $z=-1$ and $z=\frac{1}{3}$. The dominant singularity is thus $\rho=\frac{1}{3}$. Locally, we find that for some constants $c_0, c_1, c_2, \ldots$, we have

$$\begin{aligned} &\text{Expression}(z)=c_0+c_1\sqrt{1-3z}+O(1-3z) \quad \text{and} \\ &\tau\text{diff}(z)=\frac{c_2}{1-3z}+\frac{c_3}{\sqrt{1-3z}}+O(1). \end{aligned} \tag{6}$$

By virtue of general theorems to be detailed in Section 4, these local expansions can be transferred to coefficients and they provide the expansions

$$\begin{aligned} &\text{Expression}_n=\frac{c_1'}{n^{3/2}}3^n+O\left(\frac{3^n}{n}\right) \quad \text{and} \\ &\tau\text{diff}_n=c_2'3^n+\frac{c_3'}{n^{1/2}}3^n+O\left(\frac{3^n}{n}\right). \end{aligned} \tag{7}$$

Dividing these two asymptotic forms yields the main term in the statement of Automatic Theorem 1; the expansion as stated follows from suitably refined versions of (6). This completes the account of the (automatic) proof of the Automatic Theorem 1. □

The same approach will enable us to analyze a number of variants to this algorithm. For instance, by just deleting the references to the copy procedure, we obtain an algorithm that is equivalent to operating with shared pointer representations. A very similar analysis can be performed (still automatically!).

Automatic Theorem 2. *The average complexity of the differentiation algorithm applied to random expressions of size n when sharing of subexpressions is used is*

$$\frac{4}{3}n+\frac{1}{6}+O\left(\frac{1}{n}\right). \tag{8}$$

By examining Eqs. (3) and (8), we are thus able to compare two versions of the differentiation algorithm. We see that by doing (some) sharing of common subexpressions, complexity is reduced from $O(n^{3/2})$ to $O(n)$, and very precise estimates are obtained. This represents a fairly typical use of an automatic system like ΛΥΩ.

In passing, we have also obtained results about the counting of "expressions" or what amounts to the same the counting of their associated trees. Rephrasing results slightly, we have found:

Automatic Theorem 3. *The number of unary-binary trees with n nodes is the coefficient of z^n in the generating function*

$$\text{Expression}(z)=\frac{1-z-\sqrt{1-2z-3z^2}}{2z}.$$

Asymptotically, this number is

$$\text{Expression}_n = \sqrt{\frac{3}{4\pi n^3}}\,3^n - \frac{45}{32}\frac{1}{\sqrt{3\pi n^5}}\,3^n + O\left(\frac{3^n}{n^{7/2}}\right).$$

In other words, an automatic analyzer can be used for doing some amount of combinatorial counting as well. The sequence of coefficients of Expression(z) starts as 1, 1, 2, 4, 9, 21, 51, 127; these numbers are classically known as the Motzkin numbers in combinatorial theory and they appear in Sloane's book [74], under the name "generalized ballot numbers", as sequence 456.

We now propose to explain precisely on which mathematical principles such an automatic analysis can be based.

3. Algebraic analysis

The purpose of this section is to show how specifications of certain combinatorial structures together with their associated algorithmic schemes admit translations into generating functions.

We first introduce combinatorial constructions (or data type constructions, if one prefers) that form the skeleton of our system (Section 3.1). Roughly speaking, we deal with structures that are definable using products, unions, sequences, cycles, and sets (or in programming parlance, records, variable records, lists, circular lists, and unordered lists). The definitions may be either iterative (non-recursive) or recursive.

Thus, we operate with basic structures that could be termed *constructible* or *decomposable*, since they can be specified starting from basic elements by means of a fixed collection of standard set-theoretic constructions. To be more precise, the constructions operate in a parallel manner in two different universes, the "unlabelled" and the "labelled" universe—a dichotomy that is familiar from classical combinatorial analysis [38, 39, 49, 76].

An issue to be discussed is the notion of well-definedness of specifications; this is dealt with in Section 3.2. The situation there resembles that of context-free languages with respect to properness of grammatical specifications.

Next, we describe the schemes that allow us to translate combinatorial constructions into operations over generating functions (Section 3.3). The constructions that we introduce are all "*admissible*" in the sense that they translate into generating functions. Each universe is associated with its own type of generating function (either ordinary or exponential).

In Section 3.4, we introduce a collection of programming mechanisms that are in a sense the algorithmic counterpart of our standard combinatorial constructions. Intuitively, we deal with extended traversal procedures for constructible/decomposable structures. The mechanisms considered are those of selecting a component in

unlabelled	*labelled*
union [+]: union	union [+]: union
cartesian product [×]: product	partitional product [*]: product
sequence [(.)*]: sequence	partitional sequence [(.)*]: sequence
cycle [C(.)]: cycle	cycle [C(.)]: cycle
set [$\not$(.)]: set	partitional set [$\not$(.)]: set
multiset [M(.)]: multiset	

Fig. 2. The admissible constructions operate in two parallel "universes", the universe of plain unlabelled structures and that of labelled structures. The $\Lambda\gamma^{\Omega}$ forms of operators are also given (in pseudo-teletype font).

a product (record field selection), testing definitions by cases (handling records with variants), iterating over one or all components (selection/iteration) for set, cycle, or sequence constructions. This defines a closed world of algorithmic processes for which translation into complexity descriptors can be achieved automatically by means of a fixed set of rules (Section 3.5). Thus for this class, exact average-case analysis is decidable and, as it turns out, of low polynomial complexity (cf. Theorem 3.15).

3.1. Combinatorial constructions

This paper deals with discrete combinatorial models for average case analysis of algorithms. This means averaging an algorithm's cost over a class of structures (the legal inputs) of a fixed size n, using the *uniform* distribution.

Definition 3.1. A class of combinatorial structures is a pair $\langle \mathscr{C}, |\,.\,| \rangle$, where $\mathscr{C}$ is a finite or denumerable set, $|\,.\,|$ is a function from $\mathscr{C}$ to $\mathbb{N}$ called the *size* function, and for all integers n, the number of elements of $\mathscr{C}$ that have size n is finite.

We let in general $\mathscr{C}_n$ denote the subset $\{\gamma \in \mathscr{C} \,||\gamma| = n\}$. We use C_n to denote the cardinality of $\mathscr{C}_n$ and refer to the sequence of numbers $\{C_n\}_{n\geq 0}$ as the *counting sequence of the class* $\mathscr{C}$.

Typical objects that we shall consider here are words, permutations, trees and graphs of various sorts. Typically, the "size" of a word is its length, the size of a tree or a graph is the number of its nodes etc. Our main tool is going to be *generating functions* (GFs) whose definitions we now recall.

Definition 3.2. Let $\{f_n\}_{n\geq 0}$ be a sequence of numbers. We define the *ordinary generating function* (OGF) and the *exponential generating function* (EGF) of the sequence by

$$f(z) = \sum_{n\geq 0} f_n z^n \quad \text{and} \quad \hat{f}(z) = \sum_{n\geq 0} f_n \frac{z^n}{n!}. \tag{9}$$

When $\mathscr{C}$ is a class of structures with $\{C_n\}$ as its counting sequence, we call $C(z)$ and $\hat{C}(z)$ the OGF and EGF of $\mathscr{C}$. We may observe the alternate forms of GFs for structures,

$$C(z)=\sum_{\gamma\in\mathscr{C}} z^{\gamma} \quad \text{and} \quad \hat{C}(z)=\sum_{\gamma\in\mathscr{C}}\frac{z^{|\gamma|}}{|\gamma|!}. \tag{10}$$

In the sequel we stick to the notational convention of using the same groups of letters for classes of structures ($\mathscr{C}$), their counting sequences (C_n or c_n) and the corresponding generating functions ($C(z)$, $\hat{C}(z)$ or $c(z)$, $\hat{c}(z)$).

For instance, the class $\mathscr{B}$ of all binary strings is such that $\mathscr{B}_n=\{0,1\}^n$. Thus the cardinality is $B_n=2^n$, and the corresponding OGF and EGF are found to be

$$B(z)=\sum_{n\geq 0}2^n z^n=\frac{1}{1-2z} \quad \text{and} \quad \hat{B}(z)=\sum_{n\geq 0}2^n\frac{z^n}{n!}=e^{2z}. \tag{11}$$

We need a notation to go back from GFs to coefficients. If $a(z)=\sum_{n=0}^{\infty}a_n z^n$, in accordance with well established practice, we use $[z^n]a(z)$ to denote the coefficient of z^n in $a(z)$, that is to say a_n. Thus, in the notation of Eq. (9), we have

$$f_n=[z^n]f(z)=n![z^n]\hat{f}(z).$$

Abusing notations slightly, we sometimes use the convention

$$\left[\frac{z^n}{n!}\right]f(z)\equiv n![z^n]f(z).$$

In the sequel, we freely drop the "hat" in GFs whenever it is clear from context with which family we are operating. (Normally, a "universe" dictates its own choice of GFs.)

Unlabelled universe

In this universe, structures are simply composed of indistinguishable "atoms" (nodes in graphs or trees, letters in words, etc.). The size of a structure is the number of the atoms it contains. The operations allowed are

Cartesian Product, Union, Sequence, Set, Multiset, Cycle.

These operations have their usual set-theoretic meaning, except that we use a notion of "marked union" for the reason of avoiding ambiguous specifications.

The *product* relation $\mathscr{C}=\mathscr{A}\times\mathscr{B}$ means

$$\mathscr{C}=\{\gamma\in\mathscr{C}\mid\gamma=(\alpha,\beta),\alpha\in\mathscr{A},\beta\in\mathscr{B}\} \quad \text{with } |\gamma|=|(\alpha,\beta)|=|\alpha|+|\beta|. \tag{12}$$

The *union* $\mathscr{C}=\mathscr{A}+\mathscr{B}$ represents the marked union

$$\mathscr{C}=(\{\mu\}\times\mathscr{A})\cup(\{\mu'\}\times\mathscr{B}), \tag{13}$$

where μ and μ' with $\mu\neq\mu'$ are "marks" of size 0, and "$\cup$" represents the usual set-theoretic union. In other words the (marked) union defined here coincides with

the standard (set-theoretic) union whenever $\mathcal{A} \cap \mathcal{B} = \emptyset$. Otherwise, we take two disjoint copies $\mathcal{A}^\circ$, $\mathcal{B}^\circ$ of $\mathcal{A}, \mathcal{B}$ and form $\mathcal{C} = \mathcal{A}^\circ \cup \mathcal{B}^\circ$. This convention is crucial in that it eliminates all questions connected with the ambiguity of specifications.

The *sequence* class $\mathcal{C} = \mathcal{A}^*$ is defined in the usual way by

$$\mathcal{C} = \{\varepsilon\} + \mathcal{A} + (\mathcal{A} \times \mathcal{A}) + (\mathcal{A} \times \mathcal{A} \times \mathcal{A}) + \cdots. \tag{14}$$

By the *set* construction applied to $\mathcal{A}$, denoted $\mathcal{C} = \wp(\mathcal{A})$, we mean the class formed by the collection of all the *finite* subsets of $\mathcal{A}$.

$$\mathcal{C} = \wp(\mathcal{A}) \Leftrightarrow \mathcal{C} = \{\{\alpha_1, \ldots, \alpha_k\} \mid k \geq 0, \alpha_1, \ldots, \alpha_k \in \mathcal{A}\}. \tag{15}$$

The *multiset* construction **M**[.] exists only in the unlabelled universe; a multiset is a set of elements with repetitions allowed. The *cycle* construction **C**[.] applied to a set $\mathcal{A}$ is the set $\mathbf{C}(\mathcal{A})$ whose elements are (non-empty) cycles of elements from $\mathcal{A}$.

In the unlabelled universe, a *specification* of a class of combinatorial structures is a collection of (possibly recursive) equations over classes that uses only the constructors above. The initial classes are defined by the *atom* primitive that corresponds to a class only consisting of a single element (atom) of a size normally equal to 1. Thus, the class of all binary strings with alphabet $\{a, b\}$ can be specified in a non-recursive way in $\Lambda\Upsilon^{\Omega}$ format as

```
type Word = sequence(Letter);
     Letter = union(a,b);
     a, b = atom;
```

and recursively as

```
type Word = Word Letter | epsilon;
     Letter = a | b;
     a, b = atom(1); epsilon = atom(0);
```

In passing, we have illustrated some possible variations in notations: The vertical bar "|" is a synonym for union; product symbols may be omitted so that "`product(A,B)`" can be abbreviated to "`A B`"; the notation "`atom(k)`" defines an atom of size k, so that `atom` is the same as `atom(1)`, and `atom(0)` may be used to represent the empty word.

Labelled universe

The main feature here is that structures are composed of "atoms" that bear distinct integer labels from 1 to n, when the structure has size n. For instance the permutation

$$\sigma = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 2 & 4 & 5 & 6 & 3 & 1 \end{bmatrix}$$

admits the cycle decomposition $\sigma = \{(5, 3)(6, 1, 2, 4)\}$; it can be viewed as a graph whose nodes are labelled by the integers 1, 2, 3, 4, 5, 6 and with two connected components that are circular graphs.

Clearly, the standard cartesian product cannot operate directly on labelled structures (otherwise, duplicate integer labels would result). The proper notion of product[3] that is adapted to labelled structures forms pairs, but also accomplishes consistent relabellings; it is called the *partitional product*. If α and β are two labelled structures, their partitional product $(\alpha * \beta)$ is the collection of all ordered pairs $\gamma = (\alpha^{\diamond}, \beta^{\diamond})$, where $\alpha^{\diamond}$, $\beta^{\diamond}$ are obtained by performing order preserving relabellings of α, β in such a way that the resulting γ be well labelled.

The *union* operation has the same (marked) meaning in the labelled as in the unlabelled case.

Once the partitional product has been defined, the corresponding notions of *partitional sequence*, *partitional set* and *partitional cycle* constructions follow. We define the partitional sequence by

$$\mathscr{A}^* = \{\varepsilon\} + \mathscr{A} + (\mathscr{A} * \mathscr{A}) + (\mathscr{A} * \mathscr{A} * \mathscr{A}) + \cdots \tag{16}$$

and, from there, we are led to the definition of partitional power set construction, $\mathscr{C} = \not\mu(\mathscr{A})$ iff

$$\mathscr{C} = \{\{\alpha_1, \ldots, \alpha_k\} \mid (\alpha_1, \ldots, \alpha_k) \in \mathscr{A}^*, k \geq 0\}. \tag{17}$$

Cycles are defined similarly.

For instance, the class of all permutations can be defined by

```
type Permutation = set(Circular);
     Circular = cycle(Element);
     Element = Latom(1);
```

where Latom means "labelled atom". Since we operate within a labelled universe, the set and cycle constructors are implicitly to be interpreted in the labelled partitional sense.

3.2. Well defined specifications

We first need to isolate those type specifications that are well defined. Type specifications resemble context-free grammars and our problems are analogous to questions like so-called ε-freeness for context-free grammars.

More precisely, a *type specification* is an equational specification that is composed of

$$\begin{cases} \text{a set of atoms } T, \\ \text{a set of non-terminals } N, \\ \text{a set of productions } P. \end{cases}$$

The productions in P are written as equations,

$$S = \Phi(R_1, \ldots, R_k),$$

[3] This is a classical concept in combinatorial analysis. We just give here minimal definitions. The reader is referred to [24, 38, 39, 78] for background and complete definitions, or to [81] for uses in the analysis of algorithms.

where S is a non-terminal, Φ is a *constructor*, and the R_i are either atoms or non-terminals. Each non-terminal appears on the left-hand side of exactly one production.

One defines derivations, like for standard grammars: we write $A \to B$ if B derives from A in a single step and $A \overset{*}{\to} B$ if B derives from A in a sequence of steps. In this way a production can also be written $S \to \Phi$. Then, we have a precise definition of the class of structures generated by a specification.

A type specification is said to be *iterative* or *non-recursive* when the corresponding dependency graph of the productions is acyclic; otherwise, it is said to be recursive. For example, the class of non-recursive type specifications with the set of constructors {*union, product, sequence*} corresponds to regular expressions; the class of recursive type specifications with the same constructors corresponds to usual context-free grammars.

The valuation of a symbol S (atom or non-terminal) is denoted by val(S), and it is the least size of the objects generated by S (possibly ∞).

Definition 3.3. A data type specification is said to be *well defined* if it satisfies the two properties:

(1) each non-terminal has a finite valuation;

(2) for each non-terminal S, the subset $\mathcal{S}_n$ of objects of size n generated by S is finite.

Here are two instances of specifications that are not well defined. First,

```
type A = sequence(B);
     B = sequence(x);
     x = atom(1);
```

What happens is that B contains the empty structure ε, with $|\varepsilon| = 0$. Thus A contains any sequence $\varepsilon^k = (\varepsilon, \varepsilon, \ldots, \varepsilon)$ for any $k \geq 0$. Each of the ε^k has size 0, so that $A_0 = \infty$. The other example is

```
type S = S L;
     L = x | y;
     x, y = atom(1);
```

In that case, S does not specify any finite structure. In a sense, the solution for S in this equational specification is the set of *infinite* sequences, $\{x, y\}^\omega$, and we cannot assign to it any combinatorial (finite) meaning.

We naturally want to restrict attention to well defined specifications. The following proposition expresses the possibility of doing this algorithmically.

Proposition 3.4. *It is algorithmically aecidable whether a type specification is well defined or not.*

The proof is based on two lemmas that take care of each of the conditions occurring in the notion of well definedness.

Lemma 3.5. *Given a specification, the valuation of each non-terminal is computable.*

Proof. The following algorithm computes the valuation of all symbols (atoms and non-terminals) by maintaining a dynamically changing array v(.) whose final values coincide with the valuation function.

Algorithm Valuations:
for each atom a, $v(a) \leftarrow |a|$
for each non-terminal S, $v(S) \leftarrow \infty$
repeat
  for each non-terminal S **do**
    if $S \rightarrow \alpha$ where α is an atom or a non-terminal **then**
      $v(S) \leftarrow v(\alpha)$
    if $S \rightarrow union(R_1, \ldots, R_k)$ **then**
      $v(S) \leftarrow \min(v(R_1), \ldots, v(R_k))$
    if $S \rightarrow product(R_1, \ldots, R_k)$ **then**
      $v(S) \leftarrow v(R_1) + \cdots + v(R_k)$
    if $S \rightarrow \Phi(R)$ where $\Phi \in \{sequence, set, multiset\}$ **then**
      $v(S) \leftarrow 0$
    if $S \rightarrow cycle(R)$ **then**
      $v(S) \leftarrow v(R)$
  od
until v(.) is unchanged.

First this algorithm terminates because the vector v has components that lie in $\mathbb{N} \cup \{\infty\}$, which is a well ordered set, and, by construction, the vector decreases at each iteration (in the partially ordered product set). Second, after k loops, $v(S)$ is the least size of the objects derived from S in at most k steps. As v converges, the output value is the least size of all objects derived from S. □

Observe that algorithms that identify non-terminals with valuation 0 in context-free grammars are well known (cf. [40, p. 69-70] or [1]).

The next step consists of eliminating specifications with a non-terminal T such that $T_n = \infty$. We need to define a *reduced* specification as one in which: (i) all valuations $\text{val}(S)$ are finite; (ii) no production is of the form $X \rightarrow \Phi(Y)$ with Φ one of $\{sequence, multiset, cycle, set\}$ and $\text{val}(Y) = 0$. It is easy to decide using Lemma 3.5 whether a specification is reduced. Then, we have

Lemma 3.6. *Consider a reduced specification. Then the following two conditions are equivalent:*

(1) *for some integer n and non-terminal T, we have $T_n = \infty$;*

(2) *there exists a cycle $(X^{(0)}, X^{(1)}, \ldots, X^{(k)} = X^{(0)})$, such that $X^{(j)}$ appears in the production defining $X^{(j-1)}$, and for product based productions, $X^{(j-1)} \rightarrow X^{(j)}Y$ or $X^{(j-1)} \rightarrow YX^{(j)}$, we have $\text{val}(Y) = 0$.*

Proof. See Zimmermann's thesis [90]. This lemma shows that circularity (i.e., $T_n = \infty$) reduces to cycle detection in an appropriate graph. □

We can now complete the proof of Proposition 3.4 and actually derive an algorithm for deciding well defined specifications.

Algorithm Well-Definedness

(1) *Valuation*: Apply algorithm Valuations that determines all valuations (Lemma 3.5). If one valuation equals ∞, the data type specification is not well defined.

(2) *Reduction*: Check that the specification is reduced. If not, the specification is not well defined.

(3) *Circularity*: Perform the cycle detection test of Lemma 3.6 on the reduced specification.

Thus, we now know how to test algorithmically for well defined specifications. For instance, consider the following three tree specifications.

```
Ta = o | o Ta | o Ta Ta Ta;
Tb = o | w Tb | o Tb Tb Tb;
Tc = o | o Tc | w Tc Tc Tc;
o = atom(1); w = atom(0);
```

Each of the three types represents a class of unary-ternary trees, the difference being related to which nodes contribute to size and which do not. Type *Ta* is such that all nodes count, and its specification is well defined. Type *Tb* is not well defined (we can add to a tree *Tb* an arbitrary number of unary nodes without changing its size); type *Tc* is well defined in agreement with the conservation laws for trees (one cannot add ternary nodes without increasing the number of nullary nodes that contribute to size).

In the sequel, we assume that we are only dealing with well defined specifications.

3.3. Translation rules for counting generating functions

In this section, we provide translation rules from specifications to generating function equations. The main point is that the constructions that we introduced earlier (Section 3.1) are all "admissible" in the sense that they translate into generating functions.

A typical translation rule is the rule for cartesian products. Let $\mathscr{C} = \mathscr{A} \times \mathscr{B}$ be a (cartesian) product construction. By our standard convention, we let $A(z)$, $B(z)$, $C(z)$ denote the associated ordinary generating functions. Then, we have

$$C(z) = \sum_{\gamma \in \mathscr{C}} z^{|\gamma|} = \sum_{(\alpha,\beta) \in \mathscr{A} \times \mathscr{B}} z^{|\alpha|+|\beta|}$$

$$= \sum_{\alpha \in \mathscr{A}} z^{|\alpha|} \times \sum_{\beta \in \mathscr{B}} z^{|\beta|} = A(z) \cdot B(z).$$

The first sum is the definition of $C(z)$; the second sum follows from the definition of a cartesian product and of its size, cf. (12); the third sum results from distributivity.

This example shows an instance of a translation rule: *Cartesian products of sets translate into products of corresponding generating functions.* We shall present such correspondences in the form of a rule,

$$\frac{A = B \times C}{A(z) = B(z)C(z)}.$$

Unlabelled universe

In this universe, structures are simply composed of undistinguishable atoms. Translations are in terms of *ordinary* generating functions. We first list the translation rules.

Rule 1 (*Union*):

$$\frac{A = B \cup C}{A(z) = B(z) + C(z)}$$

Rule 2 (*Cartesian product*):

$$\frac{A = B \times C}{A(z) = B(z)C(z)}$$

Rule 3 (*Sequence*):

$$\frac{A = sequence(B)}{A(z) = Q(B(z))}$$

where

$$Q(f) = \frac{1}{1-f}.$$

Rule 4 (*Cycle*):

$$\frac{A = cycle(B)}{A(z) = \Phi_C(B)(z)}$$

where

$$\Phi_C(f)(z) = \sum_{k \geq 1} \frac{\phi(k)}{k} \log \frac{1}{1-f(z^k)}$$

(ϕ is the Euler totient function, that is, $\phi(k)$ is the number of positive integers not exceeding k and relatively prime to k: $\phi(1) = 1$, $\phi(2) = 1$, $\phi(3) = 2$, $\phi(4) = 2$, $\phi(5) = 4, \ldots$).

Rule 5 (*Set*):

$$\frac{A = set(B)}{A(z) = \Phi_S(B)(z)}$$

where

$$\Phi_S(f)(z) = \exp\left(\frac{f(z)}{1} - \frac{f(z^2)}{2} + \frac{f(z^3)}{3} - \cdots\right).$$

Rule 6 (*Multiset*):

$$\frac{A = multiset(B)}{A(z) = \Phi_M(B)(z)}$$

where

$$\Phi_M(f)(z) = \exp\left(\frac{f(z)}{1} + \frac{f(z^2)}{2} + \frac{f(z^3)}{3} + \cdots\right).$$

We can now state our main theorem for unlabelled structures.

Theorem 3.7. *The constructions of Union, Product, Sequence, Cycle, Set and Multiset are admissible, and their translations to ordinary generating functions are given by Rules* 1-6.

The collection of unlabelled iterative structures defines a class of generating functions that is contained in the class of elementary functions definable explicitly *from* 1, z *by application of operators* $\Omega_{\text{unlabelled}} = \{+, \times, Q, \Phi_C, \Phi_S, \Phi_M\}$.

The collection of unlabelled recursive structures defines a class of generating functions that is contained in the class of elementary functions definable implicitly *from* 1, z *by application of operators* $\Omega_{\text{unlabelled}}$.

Proof (*Indications*). The proof reduces to proving the correctness of translation Rules 1-6. This has been done already for products.

For unions, we clearly have, when $\mathscr{C} = \mathscr{A} + \mathscr{B}$:

$$\sum_{\gamma \in \mathscr{C}} z^{|\gamma|} = \sum_{\alpha \in \mathscr{A}} z^{|\alpha|} + \sum_{\beta \in \mathscr{B}} z^{|\beta|},$$

so that unions map to sums. For the sequence construction, $\mathscr{C} = \mathscr{A}^*$, since $\mathscr{C}$ is a union of products, we find that the GF $C(z)$ is a sum of products, namely:

$$C(z) = 1 + A(z) + (A(z) \cdot A(z)) + (A(z) \cdot A(z) \cdot A(z)) + \cdots = \frac{1}{1 - A(z)}.$$

The rule for the set construction is valid because $\not\mu(\mathscr{A})$ is isomorphic to an infinite cartesian product,

$$\not\mu(\mathscr{A}) \approx \prod_{\alpha \in \mathscr{A}} (\{\varepsilon\} + \{\alpha\}),$$

with ε a structure of size 0. In other words, we view a subset $\omega \subset \mathscr{A}$ as an infinite array indexed by the domain $\mathscr{A}$, where each array element is either an ε or an $\alpha \in \mathscr{A}$.

Thus, in terms of generating functions, we find

$$C(z)=\prod_{\alpha\in\mathcal{A}}(1+z^{|\alpha|})=\prod_{n=1}^{\infty}(1+z^n)^{A_n}$$

$$=\exp\left(\sum_{n=1}^{\infty}A_n\log(1+z^n)\right).$$

By using the Taylor expansion of $\log(1+x)$, we obtain

$$C(z)=\exp\left(\sum_{n\geq 1}\sum_{m\geq 1}(-1)^{m+1}A_n\frac{z^{mn}}{m}\right)$$

$$=\exp\left(\frac{A(z)}{1}-\frac{A(z^2)}{2}+\frac{A(z^3)}{3}-\cdots\right).$$

The derivation for multisets is entirely similar. It relies on the isomorphism $\mathbf{M}(\mathcal{A})\approx\prod_{\alpha\in\mathcal{A}}\{\alpha\}^*$. The translation for cycles[4] is due to Read [67].

Once the translation of basic constructions is known, the translation of complete specifications follows. Iterative specifications give rise to collections of functional equations built from 1, z by application of $\Omega_{\text{unlabelled}}$. Recursive specifications give rise to the corresponding class of functional equations. □

We observe that there are several easy extensions of these rules to slightly modified constructions. For instance, we may use

```
C = sequence(A, card>=b);
```

to construct A-sequences with at least b components. In that case, the translation to GFs is easily found to be

$$C(z)=\frac{A(z)^b}{1-A(z)}.$$

For set and cycle constructions, the corresponding functionals over GFs can be derived from Pólya's theory of counting. For instance, we have

"`C = set(A,card = 2);`" implies $C(z)=\frac{1}{2}A(z)^2+\frac{1}{2}A(z^2)$.

We use occasionally these modified rules in examples.

As an illustration of the power of this formalism, we treat a few examples giving rise to automatic theorems.

Example 3.8 (*Bracketing problems*). These problems are treated by Comtet [24, pp. 52–57] as an illustration of the technique of generating functions. Here, we are able to obtain their solutions automatically.

[4] This construction together with the set and multiset constructions can also be attached to Pólya's theory of counting [64, 65].

Automatic Theorem 4. (i) *The number of binary bracketings of a non-commutative non-associative product involving n factors is the coefficient of z^n in*

$$\mathrm{BB}(z)=\tfrac{1}{2}(1-\sqrt{1-4z}). \tag{18}$$

(ii) *The number of binary bracketings of a commutative non-associative product involving n factors is the coefficient of z^n in the function* $\mathrm{CB}(z)$ *that satisfies the functional equation*

$$\mathrm{CB}(z)=z+\tfrac{1}{2}\mathrm{CB}^2(z)+\tfrac{1}{2}\mathrm{CB}(z^2). \tag{19}$$

(iii) *The number of generalized bracketings of a non-commutative product involving n factors (where each bracket can contain two or more factors) is the coefficient of z^n in*

$$\mathrm{GB}(z)=\tfrac{1}{4}(1+z-\sqrt{1-6z+z^2}). \tag{20}$$

For part (i), there follows by the binomial expansion of $(1+x)^{1/2}$ the well known form of the *Catalan numbers*, $\mathrm{BB}_n=\frac{1}{n}\binom{2n-2}{n-1}$. For part (ii), Otter used the functional equation to prove, in 1948, that $\mathrm{CB}_n\sim 0.318(2.483)^n n^{-3/2}$ For part (iii), we shall see in the asymptotic section how to derive the asymptotic form

$$\mathrm{GB}_n=\frac{3\sqrt{2}-4}{4\sqrt{\pi n^3}}(3+\sqrt{2})^n\left(1+\mathrm{O}\left(\frac{1}{n}\right)\right). \tag{21}$$

Proof. The specifications corresponding to the three problems are

```
type BB = BB o BB | X;
     CB = X | o multiset(CB, card=2);
     GB = X | o sequence(GB, card>=2);
     X = atom(1); o=atom(0);
```

The corresponding equations for $\mathrm{BB}(z)$ and $\mathrm{GB}(z)$ are

$$\mathrm{BB}(z)=z+\mathrm{BB}^2(z) \quad \text{and} \quad \mathrm{GB}(z)=z+\frac{\mathrm{GB}^2(z)}{1-\mathrm{GB}(z)}.$$

The solutions for BB, GB then follow automatically through the resolution of algebraic equations: BB is defined directly by a quadratic equation (with a unique formal power series solution); GB satisfies an equation that reduces to a quadratic form. As to CB, its definition is (intrinsically) by means of a functional equation. □

Labelled universe

These structures are composed of atoms labelled by distinct and consecutive integers. Translations are now in terms of *exponential* generating functions. Thus, if $\mathcal{A}$ is a class of labelled structures, we operate with its EGF,

$$\sum_{n=0}^{\infty} A_n \frac{z^n}{n!},$$

which we write as $A(z)$ for notational simplicity. We again start by listing the translation rules.

Rule 7 (*Union*):

$$\frac{A = B \cup C}{A(z) = B(z) + C(z)}$$

Rule 8 (*Partitional product*):

$$\frac{A = B \times C}{A(z) = B(z)C(z)}$$

Rule 9 (*Sequence*):

$$\frac{A = sequence(B)}{A(z) = Q(B(z))}$$

where

$$Q(f) = \frac{1}{1-f}.$$

Rule 10 (*Cycle*):

$$\frac{A = cycle(B)}{A(z) = L(B(z))}$$

where

$$L(f) = \log \frac{1}{1-f}.$$

Rule 11 (*Set*):

$$\frac{A = set(B)}{A(z) = E(B(z))}$$

where

$$E(f) = \exp(f).$$

Theorem 3.9. *The constructions of Union, Product, Sequence, Cycle, Set in the labelled universe are admissible, and their translations to exponential generating functions are given by Rules* 7-11.

The collection of labelled iterative structures defines a class of generating functions that is contained into the class of elementary functions definable explicitly *from* 1, z *by application of operators* $\Omega_{\text{labelled}} = \{+, \times, Q, L, E\}$.

The collection of labelled recursive structures defines a class of generating functions that is contained into the class of elementary functions definable implicitly *from* 1, z *by application of operators* Ω_{labelled}.

Proof (*Indications*). The major point that needs justification is the rule for products. We observe that the cardinality of the partitional product $(\alpha * \beta)$ is equal to $\binom{|\alpha|+|\beta|}{|\alpha|}$. Thus, if $\mathscr{C} = \mathscr{A} * \mathscr{B}$, we find

$$C(z) = \sum_{\gamma \in \mathscr{C}} \frac{z^{|\gamma|}}{|\gamma|!} = \sum_{(\alpha,\beta)\in \mathscr{A}\times\mathscr{B}} \binom{|\alpha|+|\beta|}{|\alpha|} \frac{z^{|\alpha|+|\beta|}}{(|\alpha|+|\beta|)!}$$
$$= \sum_{\alpha\in\mathscr{A}} \frac{z^{|\alpha|}}{|\alpha|!} \times \sum_{\beta\in\mathscr{B}} \frac{z^{|\beta|}}{|\beta|!} = A(z)\cdot B(z).$$

The rule for partitional sequences follows as in the unlabelled case. The rule for sets is simpler here. From (17), a k-set is associated to $k!$ sequences (all sequences associated to a set by permuting its elements are distinct, because of the labelling of atoms!). Thus $\mathscr{C} = \mathfrak{p}(\mathscr{A})$ translates into

$$C(z) = 1 + \frac{1}{1!}A(z) + \frac{1}{2!}A^2(z) + \cdots + \frac{1}{k!}A^k(z) + \cdots = e^{A(z)}.$$

A similar reasoning gives the translation for cycles with $1/k$ replacing the factor $1/k!$. □

Labelled constructions greatly add to the expressive power of our language. We start with examples of iterative structures and then continue with recursive types. We occasionally appeal to direct variants of the rules above. For instance, for a set of cardinality larger than b, we have

$$\texttt{"A = set(B, card>b);"} \Rightarrow A(z) = e^{B(z)} - \sum_{j=0}^{b} \frac{(B(z))^j}{j!}.$$

Automatic Theorem 5. *The number β_n of partitions of a set of cardinality n into equivalence classes is the coefficient of $[z^n/n!]$ in the function*

$$\exp(e^z - 1).$$

The number of ordered partitions of a set of cardinality n into classes (where classes are ordered between themselves) is the coefficient of $[z^n/n!]$ in the function

$$\frac{1}{2 - e^z}.$$

The partition numbers are known as Bell numbers [24] and an expression can be obtained by expanding the EGF,

$$\beta_n = e^{-1} \sum_{k\geq 1} \frac{k^n}{k!},$$

a well known formula obtained by Dobiński in 1877. Ordered partitions [70, p. 99] constitute a classical example of the asymptotic analysis of meromorphic functions, and we shall derive an asymptotic form for them in the next section.

Proof. It directly results from the specification

```
type Partition = set(Block);
     Block = set(Element,card>0);
     OrderedPartition = sequence(Block);
     Element = Latom(1);  □
```

Theorem 3.10. *Given a type specification Σ, the number of arithmetic operations necessary for computing all the counting sequences associated with non-terminals up to size n is* $O(|\Sigma|n^2)$.

Proof. We consider the additional cost of computing a new coefficient $A_n = [z^n]A(z)$ where $A(z) = \Xi(B(z), C(z))$, assuming that we know already the values of A_k for $k < n$, and B_k, C_k for $k \leq n$. The proof proceeds by cases on Ξ.

Union: If $\Xi = +$, A_n is simply given by $B_n + C_n$, thus the additional cost for getting A_n is $O(1)$.

Product: If $\Xi = \times$, A_n is given by a convolution, $A_n = \sum_{k=0}^{n} B_k C_{n-k}$, and the cost is $O(n)$.

Sequence: If $A = 1/(1-B)$, we also have $A = 1 + BA$. Thus, taking coefficients, we get a recurrence[5] $A_n = [z^n](1 + BA)$, i.e., $A_n = \sum_{k=1}^{n} B_k A_{n-k}$, so that the additional cost for A_n is $O(n)$.

Labelled set: If $A = \exp(B)$, $A' = AB'$ thus $A_n = (1/n)[z^{n-1}]AB'$ and the cost is again $O(n)$.

Labelled cycle: If $A = \log(1-B)^{-1}$, we have $A' = B'/(1-B)$ or $A' = B' + BA'$. Using the corresponding recurrence, we see that the additional cost for A_n is $O(n)$.

Unlabelled set: If $A(z) = \exp(B(z) + B(z^2)/2 + \cdots)$, we have $A_n = [z^n]\exp(F(z))$, where $F(z) = B(z) + B(z^2)/2 + \cdots$. The coefficients of F are stored and computed along with the other coefficients. In this way, the cost of computing a new coefficient $[z^n]F(z)$ and the new value $A_n = [z^n]\exp(F(z))$ adds only $O(n)$ extra cost.

Unlabelled cycle: We have $A(z) = \log(1-B(z))^{-1} + \frac{1}{2}\log(1-B(z^2))^{-1} + \cdots$. We need to compute incrementally and store the coefficients of $L(z) = \log(1-B(z))^{-1}$. The additional cost of obtaining a new coefficient for $L(z)$ and for A_n is $O(n)$.

We have seen that the cost of incrementally computing one A_n is $O(n)$. Thus the total cost is $O(n^2)$.

The proof also shows that in the formula $O(n^2)$, there is an implied constant that is proportional to the size of the specification Σ. □

Note that the complexity bounds are obtained with naïve algorithms. They could be improved by using Fast Fourier Transform techniques or other classical algorithms on power series [54].

[5] This technique of forming recurrences derived from algebraic or differential equations is a familiar algorithmic trick of computer algebra.

This result, together with the companion result on the complexity of programmes, shows that exact counting results can be obtained with low computational complexity. Accordingly, this has consequences in the automatic generation of random structures in the class, since the top-down generation of structures of size n relies on these splitting probabilities. Hickey and Cohen [19] use similar techniques in order to generate words in context-free languages uniformly. Greene has given an interesting discussion of more general issues, as well as an implementation for structures definable by his "labelled grammars" [40, Ch. 4].

3.4. Programme constructions

We now introduce the class of programmes that naturally correspond to the decomposable data types that we have introduced. As we shall see, we also have translation rules into generating function equations for these schemes.

A programme consists of a *type specification* part—based on the admissible constructors described earlier—and of one or more *procedure definitions.*

Procedures are of a functional form and they are built out of a small collection of *programme constructions,* also called *programme schemes.*

The basic idea is to capture in the language a class of extended traversal procedures. Basically, we can chain operations by means of a *sequential composition* scheme. We can *test cases* for structures whose underlying type is a union of two types. We can operate on composite structures—sequences, cycles, or sets—and pick up information by either selecting a single component (*selection*) or by traversing all of them (*iteration*).

The basic operations are detailed below. Writing a procedure $P[a:A]$ specifies that the argument of P is a, and that the type of a is A.

Sequential composition. This is used for sequentially chaining operations. Our syntax here is Pascal-like and uses ";" for this scheme.

$$P[a:A] = Q[a];R[a] \quad \text{where } Q[a:A],\ R[a:A].$$

Union. For a type defined by a union, there is a test by cases.

$$A = union(B, C)$$
$$P[a:A] = \textbf{if } a \text{ } ? \textbf{ then } Q[a] \textbf{ else } R[a]$$
where $Q[b:B]$, $R[c:C]$.

Product. The selection scheme descends into one component of a product:

$$A = product(B, C)$$
$$P[(b, c):A] = Q[b]$$
where $Q[b:B]$.

Sequence, Cycle, Set and Multiset. The *selection* scheme extracts one component (a fixed component like the first one for a sequence, and a random component for

a cycle or a set). For instance, we have

$A = sequence(C)$
$P[a:A] = Q[a[1]]$
where $Q[c:C]$.

The corresponding *iteration* scheme examines all components:

$A = cycle(C)$
$P[a:A] =$ **forall** b **in** a **do** $Q[b]$
where $Q[c:C]$.

These schemes exist in parallel in the unlabelled and the labelled universe, with the obvious restrictions: the notion of product is that of the universe under consideration; multisets are distinguished from sets only in an unlabelled universe.

Observe that there are no explicit variable assignments, and in a deep sense, *one cannot modify structures nor create new structures.* Operations are thus in essence limited to traversal procedures.[6] However, as we shall see, many algorithms that do modify their data can be emulated in the language.

The concrete syntax that we use in examples is an incarnation of the abstract schemes above; it should be self-explanatory; see the example of symbolic differentiation discussed in Section 2.

For programmes, there is a notion of well-definedness that is analogous to that of type specifications.

Definition 3.11. A programme over a well defined type specification is itself *well defined* iff for each procedure Q and each input x, the execution of Q on x terminates in a finite number of steps.

The situation is made easy here since all programmes satisfy a descent property: Given a procedure call $Q(x)$, all the calls $R(y)$ that are generated operate with y's that are substructures of x. The only way a procedure Q can loop is therefore by generating a call sequence,

$$Q(x) \to Q_1(x) \to \cdots \to Q_l(x) \to Q(x),$$

with a stationary argument. Such a property is in fact syntactically decidable. (See our earlier discussion about circularity in data type specifications and Zimmermann's thesis [90] for details.)

[6] Judging from the entirety of the analyses contained in Knuth's volume on sorting and searching [53], the only algorithms that we know how to analyze are those whose complexity is equivalent to a parameter of a *static structure.* No general method is known in order to analyze intrinsically *dynamic* algorithms that repeatedly modify a structure. Examples that typically leave us helpless are heapsort and balanced trees that modify either an ordered array structure or a tree structure. The reader can consult the classic paper of Jonassen and Knuth [48] to see what awaits the analyst confronted with such problems, when the size is restricted to $n = 3!$ Thus, the limitation under discussion is not an essential bottleneck, given our current state of knowledge in the analysis of algorithms.

Proposition 3.12. *It is algorithmically decidable if a programme is well-defined or not.*

3.5. Complexity descriptors

The *ordinary complexity descriptor* of a procedure P with input in a set $\mathscr{A}$ is the generating function

$$\tau P(z) = \sum_{a \in \mathscr{A}} \tau P\{a\} z^{|a|}.$$

The *exponential complexity descriptor* is

$$\widehat{\tau P}(z) = \sum_{a \in \mathscr{A}} \tau P\{a\} \frac{z^{|a|}}{|a|!}.$$

There, $\tau P\{a\}$ denotes the complexity (cost) of procedure P applied to input a. The complexity is always represented by the number of times some explicitly designated operations (basic procedure calls) are performed. On our examples, this is represented by the "measure" directive.

The rules that follow enable us to translate programme schemes into functional equations over complexity descriptors. The complexity descriptors and counting GFs appearing there are to be taken as either ordinary (in an unlabelled universe) or exponential (in a labelled universe); in the latter case, we omit the "^" token of EGFs. With this convention some of the rules can be grouped together: for instance, the rule for sequences is to be understood as a rule for ordinary GFs and complexity descriptors in the unlabelled case, and to be interpreted as a rule for exponential GFs in the labelled case.

Whenever we need to emphasize that the complexity of P is taken with respect to inputs in A, we write $\tau P_{\downarrow A}(z)$ instead of $\tau P(z)$.

Rule 12 (*Elementary costs*): This corresponds to a cost measure that is declared to assign constant cost μ to each procedure call Q(.).

$$\frac{P[a:A] = Q[a] \quad \tau Q\{a\} \equiv \mu}{\tau P(z) = \mu \cdot A(z)}$$

Rule 13 (*Sequencing*):

$$\frac{P[a:A] = Q[a]; R[a]}{\tau P(z) = \tau Q(z) + \tau R(z)}$$

Rule 14 (*Union*):

$$\frac{A = union(B, C) \quad P[a:A] = \textbf{if } a \in B \textbf{ then } Q[a] \textbf{ else } R[a]}{\tau P_{\downarrow A}(z) = \tau Q_{\downarrow B}(z) + \tau R_{\downarrow C}(z)}$$

Rule 15 (*Product*):

$$\frac{A = product(B, C) \quad P[a:A] = Q[a[1]]}{\tau P_{\downarrow A}(z) = \tau Q_{\downarrow B}(z) C(z)}$$

Rule 16 (*Sequence*: *selection*):

$$\frac{A = sequence(B) \quad P[a:A] = Q[a[1]]}{\tau P(z) = \tau Q(z)/(1 - B(z))}$$

Rule 17 (*Sequence*: *iteration*):

$$\frac{A = sequence(B) \quad P[a:A] = \textbf{forall } b \textbf{ in } a \textbf{ do } Q[b]}{\tau P(z) = \tau Q(z)/(1 - B(z))^2}$$

Rule 18 (*Set*: *selection*): In an unlabelled universe,

$$\frac{A = set(B) \quad P[a:A] = Q[a[1]]}{\tau P(z) = \sum_{n=1}^{\infty} \tau Q_n z^n \left[\int_0^1 \frac{B^{\mathbf{S}}(z,u)}{1+uz^n}\, du \right]}$$

where $B^{\mathbf{S}}(z,u) = \prod_{b \in B} (1 + uz^{|b|})$. In a labelled universe,

$$\frac{A = set(B) \quad P[a:A] = Q[a[1]]}{\tau P(z) = (\exp(B(z)) - 1)/B(z)\tau Q(z)}$$

Rule 19 (*Set*: *iteration*): In an unlabelled universe,

$$\frac{A = set(B) \quad P[a:A] = \textbf{forall } b \textbf{ in } a \textbf{ do } Q[b]}{\tau P(z) = \Phi_S(B)(z)(\tau Q(z) - \tau Q(z^2) + \tau Q(z^3) - \cdots)}$$

In a labelled universe,

$$\frac{A = set(B) \quad P[a:A] = \textbf{forall } b \textbf{ in } a \textbf{ do } Q[b]}{\tau P(z) = \exp(B(z))\tau Q(z)}$$

Rule 20 (*Multiset*: *selection*): In an unlabelled universe,[7]

$$\frac{A = multiset(B) \quad P[a:A] = Q[a[1]]}{\tau P(z) = \sum_{n=1}^{\infty} \tau Q_n z^n \left[\int_0^1 \frac{B^{\mathbf{M}}(z,u)}{1-uz^n}\, du \right]}$$

where $B^{\mathbf{M}}(z,u) = \prod_{b \in B} 1/(1 - uz^{|b|})$.

Rule 21 (*Multiset*: *iteration*): In an unlabelled universe,

$$\frac{A = multiset(B) \quad P[a:A] = \textbf{forall } b \textbf{ in } a \textbf{ do } Q[b]}{\tau P(z) = \Phi_M(B)(z)(\tau Q(z) + \tau Q(z^2) + \tau Q(z^3) + \cdots)}$$

Rule 22 (*Cycle*: *selection*): In an unlabelled universe,

$$\frac{A = cycle(B) \quad P[a:A] = Q[a[1]]}{\tau P(z) = \sum_{k \geq 1} \frac{\phi(k)}{k} \log \frac{1}{1 - B(z^k)} \frac{\tau Q(z^k)}{B(z^k)}}$$

In a labelled universe,

$$\frac{A = cycle(B) \quad P[a:A] = Q[a[1]]}{\tau P(z) = \log \frac{1}{1 - B(z)} \frac{\tau Q(z)}{B(z)}}$$

[7] The *multiset* constructor differs from the *set* constructor only in an unlabelled universe.

Rule 23 (*Cycle*: *iteration*): In an unlabelled universe,

$$\frac{A = cycle(B) \quad P[a:A] = \textbf{forall } b \textbf{ in } a \textbf{ do } Q[b]}{\tau P(z) = \sum_{k \geq 1} \phi(k) \frac{1}{1 - B(z^k)} \tau Q(z^k)}$$

In a labelled universe,

$$\frac{A = cycle(B) \quad P[a:A] = \textbf{forall } b \textbf{ in } a \textbf{ do } Q[b]}{\tau P(z) = \frac{1}{1 - B(z)} \tau Q(z)}$$

That these rules are correct follows from techniques akin to those employed in proving the corresponding results for counting generating functions,[8] the cases of the unlabelled set/multiset/cycle constructions being trickier. We shall refer to Zimmermann's thesis [90] for detailed proofs.

Theorem 3.13. (i) *For each of the four data type classes—unlabelled iterative, labelled iterative, unlabelled recursive, labelled recursive—the corresponding class of programme schemes translates into functional equations over complexity descriptors.*

(ii) *The complexity descriptors of programmes operating on a collection of labelled iterative structures are definable explicitly from* 1, z *by application of operators* $\Omega^*_{\text{labelled}} = \{+, \times, Q, E, L, E^*, L^*\}$, *where*

$$E^*(f) = (E(f) - 1)/f, \qquad L^*(f) = L(f)/f.$$

(iii) *The complexity descriptors of programmes operating on a collection of labelled recursive structures satisfy systems of linear equations whose coefficients are definable from* 1, z *by application of operators* $\Omega^*_{\text{labelled}}$.

For unlabelled types, in general, we obtain functional equations involving the Φ operators of type constructions and a class of operators Ψ associated with selection and iteration on sets, multisets and cycles.

Example 3.14 (*Cycles in a random permutation*). Our first example is a programme that counts the number of cycles in a permutation.

Automatic Theorem 6. (i) *The expected number of cycles in a random permutation of* n *elements is equal to the coefficient of* $[z^n]$ *in the generating function*

$$H(z) = \frac{1}{1 - z} \log \frac{1}{1 - z}. \tag{22}$$

[8] In a sense, the rules for data types are reduced (homomorphic) images of the type specifications themselves. The translation to complexity descriptors instead resembles a generalized derivation on type specifications.

(ii) *This expected number has the asymptotic form*

$$[z^n]H(z) = \log n + \gamma + \frac{1}{2n} - \frac{1}{12n^2} + \frac{1}{120n^4} + O\left(\frac{1}{n^6}\right), \tag{23}$$

where $\gamma \approx 0.57721$ *is the Euler constant.*

The first part is equivalent to the well known assertion that the mean number of cycles is given by a harmonic number,

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}.$$

The "automatic" character of part (ii) results from the developments of the next section.

Proof. The type specification of permutations has been studied already in Section 3.1.

```
type Permutation = set(Circular);
     Circular = cycle(Element);
     Element = Latom(1);
```

To count the number of cycles in a permutation, it suffices to traverse the permutation and on each cycle trigger a procedure, count, which does nothing but whose cost is declared to be equal to 1:

```
procedure CountCycles(p : Permutation);
begin
  forall c in p do
    count(c);
end;

measure count : 1;
```

The type specifications lead to a collection of equations for counting generating functions.

$$\begin{aligned} &\text{Permutation}(z) = \exp(\text{Circular}(z)) \\ &\text{Circular}(z) = \log(1 - \text{Element}(z))^{-1} \\ &\text{Element}(z) = z. \end{aligned} \tag{24}$$

Turning to procedures, the rule for initial costs gives

$$\tau\text{count}(z) = \text{Circular}(z) = \log\frac{1}{1-z}; \tag{25}$$

the rule for set iteration provides

$$\tau\text{CountCycles}(z) = \tau\text{count}(z) \cdot \text{Permutation}(z) \tag{26}$$

which is equivalent to the first assertion of the theorem, with $H(z)$ being an abbreviation for τCountCycles(z). □

Finally, these (often huge!) generating function equations also convey some meaning, as the following theorem shows.

Theorem 3.15. *Given a programme specification Π, the average cost of its procedures operating on uniform random inputs of size up to n can be determined in* $O(|\Pi| n^3 \times \log n)$ *arithmetic operations.*

The proof proceeds along the lines of that of data types which we have given in sufficient detail.

3.6. Examples

We collect here a few more examples meant to illustrate the expressive power of our formalism. In order not to make the statements too cumbersome we sometimes directly cite an asymptotic result. In that case, it is to be understood that the asymptotic part of the proof will follow from the developments given in the next section.

Example 3.16 (*Denumerants*). In how many ways can one attain a total of n centimes, with coins of denominations 1, 2 and 3 centimes? The problem is one of special integer partition counting [24, p. 108].

Automatic Theorem 7. *The number of partitions of n into summands equal to* 1, 2, *or* 3 *is*

$$\tfrac{1}{12}n^2+\tfrac{1}{2}n+O(1).$$

Proof. The formal description of the problem is the following.

```
type Sum = multiset(Coin);
     Coin = one | two | three;
     one = atom(1); two = atom(2); three = atom(3);
```

According to rules 1 and 6, the ordinary generating function is computed explicitly from this description.

$$\mathrm{Sum}(z)=\frac{1}{(1-z)(1-z^2)(1-z^3)}$$

From this generating function, and with theorems of the following section, one deduces the asymptotic expansion given above. □

Example 3.17 (*Compositions*). A 1-2-composition of n is a sequence of integers $(i_1, \ldots, i_k)$ in $\{1, 2\}$ whose sum equals n.

Automatic Theorem 8. *The number of* 1-2-*compositions of n is asymptotically*

$$\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1}+O\left(\left(\frac{1+\sqrt{5}}{2}\right)^{n}\Big/ n\right).$$

Proof. A composition is simply a list of summands

```
type Composition = sequence(Summand);
     Summand = one | two;
     one = atom(1); two = atom(2);
```

According to rules 1 and 3, the ordinary GF for 1-2-compositions is thus

$$\text{Composition}(z) = \frac{1}{1-(z+z^2)}.$$

The coefficient of z^n is a Fibonacci number. The well known asymptotic expansion follows automatically from the algorithms of Section 4. □

Example 3.18 (*Path length in trees*). Our problem here is to analyze path length in general plane trees where all node degrees are allowed. The programme operates on an unlabelled, recursively defined type. Its specification closely mimics the inductive definition of path length: if $t = (o, t_1, \ldots, t_k)$, then

$$\pi[t] = |t| + \pi[t_1] + \cdots + \pi[t_k].$$

Automatic Theorem 9. *The expected path length in a general plane tree with n nodes is asymptotically*

$$\tfrac{1}{2}\sqrt{\pi}n^{3/2} + \tfrac{1}{2}n + O(n^{1/2}).$$

Proof. It suffices to translate the classical inductive definition of path length into our framework.

```
type Tree = Node sequence(Tree);
     Node = atom(1);

procedure Size(t : Tree);
begin
  count;
  case t of
    (root,subtrees): forall u in subtrees do
                       Size(u);
  end;
end;

procedure PathLength(t : Tree);
begin
  Size(t);
  case t of
    (root,subtrees): forall u in subtrees do
                       PathLength(u);
  end;
end;

measure count : 1;
```

From rules 2 and 3, we obtain the counting GF for trees

$$\text{Tree}(z) = \frac{1-\sqrt{1-4z}}{2}.$$

From rules 12, 13, 15 and 17, the complexity descriptor for the procedure PathLength is obtained (before simplification) by the $\Lambda\Upsilon\Omega$ system as

$$\tau\text{PathLength}(z) = \frac{(\sqrt{1-4z}-1)(1+\sqrt{1-4z})^4}{16+8\sqrt{1-4z}-96z+8(1-4z)^{3/2}+128z^2-32z\sqrt{1-4z}}.$$

From these two GFs, the result follows by asymptotic expansion of coefficients. □

Example 3.19 (*Derangements and singleton cycles in permutations*). We want to prove here the assertions made in the introduction regarding fixed points in permutations.

Automatic Theorem 10. *The number of derangements* (*permutations without fixed point*) *of* 1..*n is equal to*

$$D_n = n! \cdot [z^n]\frac{e^{-z}}{1-z}.$$

The expected number of singleton cycles (*cycles of size* 1) *in a random permutation of n is equal to* 1.

Proof. The formal description of derangements and singleton cycles follows, and the procedure FixedPoints counts the number of singleton cycles in a permutation.

```
type Derangement = set(Circular2);
     Circular2 = cycle(Node,card>=2);
     Node = Latom(1);
     Permutation = set(Singleton | Circular2);
     Singleton = Latom(1);

procedure FixedPoints(p : Permutation);
begin
  forall c in p do
    case c of
      Singleton : count
    end;
end;

measure count : 1;
```

According to rules 10, 11, 12 and 19, the counting GF for the number of derangements and the complexity descriptor of the procedure are

$$\text{Derangement}(z) = \frac{\exp(-z)}{1-z}, \qquad \tau\text{FixedPoints}(z) = \frac{z}{1-z},$$

from which both statements of the theorem result immediately. □

Example 3.20 (*Cyclic points in random mappings*). We consider random mappings from a finite set to itself with the special property that each point has either 0 or 2 antecedents. This is an approximate model for quadratic functions $x \mapsto x^2 + a \pmod{n}$. Such a mapping is equivalent to a binary functional graph, i.e., a digraph in which each point has outdegree 1 and indegree 0 or 2. A binary functional graph can be specified in our formalism; it is a set of connected components; each component has a unique cycle; on each point of the cycle are planted binary trees. For motivations related to cryptography and random number generation (see, e.g., [31]), our goal is to determine the average proportion of points that lie on a cycle (*cyclic points*).

```
type Mapping = set(Component);
     Component = cycle(PlantedTree);
     PlantedTree = node Tree;
     Tree = node | node set(Tree,card=2);
     node = Latom(1);

procedure CountCyclicPoints (m : Mapping);
begin
  forall c in m do
    CountCyclicPointsInComponent(c);
end;

procedure CountCyclicPointsInComponent (c : Component);
begin
  forall t in c do
    count;
end;

measure count : 1;
```

Automatic Theorem 11. *The average number of cyclic points in a random binary functional graph of n points is for* $n \equiv 0 \pmod 2$

$$\frac{[z^n]\dfrac{1-\sqrt{1-2z^2}}{1-2z^2}}{[z^n]\dfrac{1}{\sqrt{1-2z^2}}}.$$

Proof. The exponential GF for binary functional mappings and the complexity descriptor for the number of cyclic points in such mappings are found to be

$$\mathrm{Mapping}(z) = \frac{1}{\sqrt{1-2z^2}},$$

$$\tau\mathrm{CountCyclicPoints}(z) = \frac{1-\sqrt{1-2z^2}}{1-2z^2}. \qquad \square$$

This asymptotic result is of some relevance to the analysis of an integer factorization algorithm due to Pollard, the so-called Pollard *rho-method* (see, e.g., [31]).

4. Asymptotic analysis of a class of elementary functions

At this stage, the algebraic theory of generating functions—at least in cases where explicit solutions exist—provides an expression of a function in terms of basic operators associated with combinatorial constructions. In this way, we are confronted with the problem of estimating coefficients of generating functions of rather diverse and complicated forms.

Apart from the simplest cases (like the GF of Fibonacci numbers which we encountered when analyzing 1-2 compositions), no "closed form" for the coefficients is available in general. However, it appears that a considerable amount of asymptotic information on the coefficients of a GF $f(z)$ is contained in the *singularities* of $f(z)$, itself viewed as an *analytic function* of the *complex* variable z. For the automatic extraction process that we envision, we must also render the method free from analysis, and purely formal or "algebraic". This is made possible by the approach explained here. The end result is quite simple. For a large class of functions f arising from combinatorial enumerations, the nth Taylor coefficient $f_n \equiv [z^n]f(z)$ has an asymptotic form,

$$f_n \sim C\rho^{-n}n^s(\log n)^k, \tag{27}$$

with k an integer, C, ρ, s real numbers. All the quantities appearing in the estimate (27) are algorithmically computable.

A sample of functions related to combinatorial enumerations that we discuss throughout this section is the following.

$$f_0(z) = \frac{1}{1-z-z^2}, \qquad f_1(z) = \frac{1}{2-e^z}, \qquad f_2(z) = \frac{1}{1-\log[1/(1-z-z^2)]},$$

$$f_3(z) = \frac{1}{\sqrt{1-z}}\,\frac{1}{1-z^3},$$

$$f_4(z) = \log\frac{1}{1-z\log[1/(1-z^2)]} + \frac{1}{(1-z^3)^5} + \exp(z\,e^z),$$

$$f_5(z) = \exp\left(z^2\log\frac{1}{1-z^4}\right) + \frac{1}{1-2z^6},$$

$$f_6(z) = \exp\left(z\log\frac{1}{1-4z^4}e^{z/(1-z^3)}\right) + e^z\log\frac{1}{1-2z^2} + 1 + z,$$

$$f_7(z) = \exp\left(\log\frac{1}{1-z-z^2}e^{z/(1-z)}\right).$$

In this section, we propose an algorithm that operates on explicitly defined functions of which $f_0, \ldots, f_7$ are typical. All these examples have singularities at a finite distance. When analyzing such a function $f(z)$, the following strategy is used.

(1) The analytic properties of f are first introduced into the game by means of the *Cauchy coefficient formula*

$$f_n \equiv [z^n]f(z) = \frac{1}{2i\pi}\oint f(z)\frac{dz}{z^{n+1}}. \tag{28}$$

(2) It is well known from the theory of Cauchy's formula (28), see [80], that the *singularities* of a function nearest to the origin determine the radius of convergence of the function (i.e., the series defining the function). Such singularities are known as *dominant* singularities and the discussion above reduces to the assertion: *the modulus of the dominant singularities of an analytic function $f(z)$ gives the radius of convergence of the series form of $f(z)$*. Then, if we let ρ denote the radius of convergence of $f(z)$, the coefficients $\{f_n\}_{n\geq 0}$ satisfy the basic relation

$$\limsup_{n\geq 0}|f_n|^{1/n} = \frac{1}{\rho}. \tag{29}$$

This property is often written in a more suggestive way as an approximation relation

$$f_n \approx \rho^{-n} \quad \text{where} \quad \rho = \min\{|z| \mid f(z) \text{ is singular}\}.$$

The precise meaning of the formula $f_n \approx \rho^{-n}$ is that $f_n \sim \rho^{-n}\omega(n)$, where $\omega(n)$ satisfies $\limsup|\omega(n)|^{1/n} = 1$, i.e., the growth of ω is slower than any increasing exponential but faster than any decreasing exponential infinitely often. The formula $f_n \approx \rho^{-n}$ thus indicates that ρ^{-n} captures the main exponential growth of f_n.

For instance, the dominant singularity of $f_1(z)$ is at $\rho = \log 2$ which cancels the denominator. The dominant singularity of $f_2(z)$ can be determined by looking at places where either the logarithm becomes singular or the denominator cancels. In this fashion, we obtain the approximate formulae

$$[z^n]\frac{1}{2-e^z} \approx \left(\frac{1}{\log 2}\right)^n, \qquad [z^n]\frac{1}{1-\log[1/(1-z-z^2)]} \approx \left(\frac{-1+\sqrt{5-4e^{-1}}}{2}\right)^{-n}.$$

(3) The modulus of the dominant singularities of $f(z)$ thus provides the first level of information on coefficients of a function, in the approximate form of an exponential term. If a function has a unique dominant singularity, this is usually enough to conclude the analysis by local singularity analysis, as explained below.

However, some functions hide *periodicities* in the behaviour of their coefficients. For instance the GF $f_3(z)$ expands as

$$1.0+0.50z+0.38z^2+1.3z^3+0.77z^4+0.62z^5+1.5z^6+0.98z^7+0.82z^8$$
$$+1.7z^9+1.2z^{10}+0.99z^{11}+1.9z^{12}+1.3z^{13}+1.1z^{14}+2.0z^{15}+O(z^{16}).$$

Plotting the values of these coefficients (see Fig. 3) suggests that the coefficients go by groups of three. It is indeed the case, and this is due to the presence of three dominant singularities, namely

$$1, \quad 1 \cdot e^{2i\pi/3}, \quad 1 \cdot e^{-2i\pi/3},$$

these singularities being related to the presence of $(1-z^3)$ in the denominator. Therefore, one of the major problems, whenever periodicities arise, consists of determining the directions where the dominant singularities lie. These directions are called *dominant directions*. The process of analyzing the coefficients of a function is shown to decompose into a finite collection of aperiodic problems of a simpler form.

(4) Leaving apart the periodicity phenomena—this is possible either because the function to analyze $f(z)$ has no periodicities, or because $f(z)$ has already been decomposed—the problem is thus to quantify the subexponential factor $\omega(n)$ in the formula $f_n \sim \rho^{-n}\omega(n)$.

It turns out that there is a correspondence between the *singular rates of growth* of functions around their singularities and the asymptotic (subexponential) rates of growth of their coefficients. Here are a few examples of the correspondence, for functions singular at 1,

$$\frac{1}{\sqrt{1-z}} \mapsto \frac{1}{\sqrt{\pi n}}, \quad \frac{1}{1-z}\log\frac{1}{1-z} \mapsto \log n,$$

$$\exp\left(\frac{z}{1-z}\right) \mapsto \frac{e^{2\sqrt{n}}}{2(\pi e)^{1/2} n^{3/4}}.$$

(The last transformation belongs to the theory of saddle point integrals which we discuss in Section 5.5.)

(5) In our approach, the problem of finding the asymptotic growth of coefficients of $f(z)$ reduces to determining locally the behaviour of the function around all its dominant singularities. If the collection of these is $\{\sigma_j = \rho e^{i\theta_j}\}_{j\in J}$, for a finite index set J and real angles θ_j, then, under normal circumstances, by recomposing elements of the form $\sigma^{-n}\omega(n)$, we obtain

$$f_n \sim \frac{1}{\rho^n} \sum_{j\in J} \omega_j(n)\, e^{in\theta_j},$$

where the $\omega_j(n)$ are functions of subexponential growth.

Fig. 3. A graph of the coefficients of z^n in $f_3(z)$, as a function of n.

The general principle that guides us is that functions arising from the automatic algebraic construction of generating functions have coefficients that are also automatically analyzable. More will be said to support this broad claim in Sections 5 and 6.

We now propose to implement this programme in detail on the class $\mathscr{E}$ of *elementary functions* that appear as generating functions of well defined labelled iterative structures. (This class should be called in full the class of LI-elementary functions.)

Definition 4.1. The class of elementary functions $\mathscr{E}$ is defined as the class of functions containing the monomials 1, z and closed under the operations of $\Omega_{\text{labelled}} = \{+, \times, Q, L, E\}$, where

$$Q(f) = \frac{1}{1-f}, \qquad L(f) = \log \frac{1}{1-f}, \qquad E(f) = \exp(f),$$

with the further restriction that all operations take place in the ring of formal power series $\mathbb{Q}[[z]]$.

The requirement that operations be formal means that Q, L and E can only be applied to functions f such that $f(0) = 0$, a restriction which is satisfied exactly by those generating functions that arise from *well defined* specifications in the sense of algebraic enumerations (Section 3).

The restriction to Ω and $\mathbb{Q}$ is not strict. The algorithms we shall develop apply almost verbatim to enriched classes, where we allow modified operators like $E^*(f) = (e^f - 1)/f$ etc. Thus, though we state propositions for $\mathscr{E}$, trivially amended results hold true for larger classes, from which we occasionally borrow examples such as f_1 or f_3. The remaining functions $f_0, f_2, f_4, f_5, f_6, f_7$ all belong to $\mathscr{E}$, and thus they are generating functions of some elementary (i.e., labelled iterative) structures.

There is an important subclass of the elementary class $\mathscr{E}$—the class $\mathscr{E}_{\text{AL}}$ of "algebraic-logarithmic" functions—for which a complete asymptotic analysis of coefficients can be developed *automatically* by means of the strategy that we have exposed. It is our purpose now to explore properties of the class $\mathscr{E}$ and its distinguished subclass, and to illustrate our general philosophy by working out the algorithms in some detail.

The programme presented in the next paragraphs can be outlined as follows:

Algorithm Equivalent

Input: A function f from the elementary class $\mathscr{E}$.

Output: An asymptotic form of $[z^n]f(z)$ when f is in a proper subclass, $f \in \mathscr{E}_{\text{AL}}$.

(1) *Radius*: Find the radius of convergence ρ of f.

(2) *Directions*: Compute the dominant directions.

(3) *Expansion*: Determine the growth of the function about its dominant singularities.

(4) *Transfer*: Deduce the growth of the coefficients.

The labels Radius, Directions, Expansion, Transfer refer to specific algorithms.

Other methods, like saddle-point analysis, that are operational for dealing with functions of "violent" singular growth, like entire functions, are discussed in Section 5.3.

4.1. Dominant singularities and principal exponential growth

Our first result shows that the dominant exponential growth $f_n \approx \rho^{-n}$ is computable for all f in the class $\mathscr{E}$. We start to exploit the analytic fact that functions in $\mathscr{E}$ have positive coefficients (they are GFs) and rely on structural induction.

Proposition 4.2. *Let $f(z)$ be an elementary function in the class $\mathscr{E}$. Then*

(1) *the radius of convergence ρ of f satisfies $0 < \rho \leq \infty$;*

(2) *it is decidable whether f is entire or not, i.e., whether $\rho = \infty$ or $\rho < \infty$;*

(3) *whenever $\rho < \infty$, ρ is a dominant singularity of f; furthermore, function f is infinite at ρ, which means that $f(x) \to +\infty$ as $x \to \rho$ from the left;*

(4) *the radius of convergence ρ of f, when it is finite, is computable to any precision $\varepsilon > 0$.*

Proof. (1) These functions are analytic at the origin because they are either polynomials or compositions of exp, L, Q, with functions that are 0 at 0. Hence by induction they are analytic at 0.

(2) By induction it is easy to see that f is entire if and only if neither Q nor L appear in its expression.

(3) This is a special case of Pringsheim's theorem [80]. Since the coefficients of our functions are positive, using the triangular inequality shows that they are maximal along the real axis. That they are infinite at their singularity follows again by induction since singularities in this class can only arise by Q or L.

(4) The computation of ρ is done by the following algorithm:

Algorithm Radius

Input: An expression $f \in \mathscr{E}$.

Output: The radius of convergence of f within a fixed accuracy $\varepsilon > 0$.

(1) If f is a polynomial, then its radius of convergence is infinite.

(2) If f is $\exp(g)$ then its radius is that of g.

(3) If f is $Q(g)$ or $L(g)$, then the radius is the smallest real positive root of $g(x) = 1$.

(4) If f is a sum or a product, then its radius is the minimum of those of its arguments.

The main observation is relative to Step 3: $g(x) = 1$ has a single root in the interval $]0, \rho[$ with ρ the radius of convergence of g. This root can be computed to any accuracy by classical numerical algorithms. □

Example 4.3. We consider the function f_4 defined by

$$f_4(z) = \log \frac{1}{1 - z \log[1/(1-z^2)]} + \frac{1}{(1-z^3)^5} + \exp(z\, e^z).$$

Although the smallest real singularity ρ of f_4 cannot be expressed in closed form, it is not difficult to see (automatically by the above algorithm, or by hand) that ρ is the smallest singularity of the outer logarithm and to compute an approximate value

$$\rho \approx 0.835408159 = \min\left\{0 < x < 1 \,\middle|\, x \log \frac{1}{1-x^2} = 1\right\}.$$

4.2. Dominant directions and periodicities

In order to take into account the periodicities that may occur in coefficients of functions, we introduce the *reduced form* of a function. The reduced form of f is a triple (a, g, p) such that

$$f(z) = z^a g(z^p),$$

with g satisfying $g(0) \neq 0$, p and a two integers $0 \leq a < p$, and $p \geq 1$ as large as possible. The number p is called the *period* of f.

Observe also that the period p is visible on the Taylor coefficients of f: the indices of the non-zero coefficients of f are included in a unique arithmetic progression of ratio p,

$$\{n \mid f_n \neq 0\} \subseteq \{a + jp\}_{j \geq 0}.$$

A function that has a period $p \geq 2$ is said to be *purely periodic.* For instance, any odd or even function is purely periodic.

Proposition 4.4. *A function f in $\mathscr{E}$ has a reduced form $f(z) = z^a g(z^p)$, with g belonging to $\mathscr{E}$. The quantities a, p, g are effectively computable.*

Proof. Both parts of the proof are consequences of the following algorithm.

Algorithm Reduction
Input: $f \in \mathscr{E}$.
Output: A pair (a, p) and a function g.

(1) If f is a polynomial, expand it, take p as the gcd of the differences of the exponents of the monomials, and a as the smallest exponent modulo p.

(2) If f is $Q(g)$, $L(g)$, or $\exp(g)$, apply the algorithm to its argument, then take the gcd of a and p as the new p while a becomes 0.

(3) If f is a sum or a product, apply the algorithm to its components, and take the proper gcds.

All these operations are purely syntactical, and one can check that the final g appearing in the reduced form is always a function of $\mathscr{E}$. Full details will appear in [73]. □

This algorithm will serve to compute all the dominant singularities of a function. But we first need to introduce an oracle which plays an important role.

Definition 4.5. We call $\mathcal{O}_1$ the following oracle: Given two functions g and h in $\mathscr{E}$ that are 0 at the origin, with r_g and r_h the smallest real positive roots of $g(x)=1$ and $h(x)=1$, the oracle outputs one of

$$r_g < r_h, \qquad r_g = r_h, \qquad r_g > r_h.$$

The oracle enables us to state the following proposition.

Proposition 4.6. *The dominant directions of a function $f \in \mathscr{E}$ are computable conditionally upon oracle $\mathcal{O}_1$. These directions are all commensurable with π.*

Proof. It operates by a structural induction that we embody in the algorithm Directions.

Algorithm Directions
Input: $f \in \mathscr{E}$.
Output: A set of angles.

(1) If f is $\exp(g)$ then apply the algorithm to g.

(2) If f is $Q(g)$ or $L(g)$ then apply Reduction to reduce $g(z)$ into $z^a h(z^p)$; set $q = \gcd(a, p)$ and return $\{(2k\pi/q),\ k = 1..q\}$.

(3) If $f = g + h$ or $f = g \cdot h$, then first use algorithm Radius to compute the radii of convergence of g and h, then use the oracle $\mathcal{O}_1$ to compare them; if they are different then apply Directions on the function with the smallest one, otherwise apply it to both of them and return the union of their dominant directions. □

We observe that the corresponding problems are well worked out in the case of rational series [11, 27, 71].

Example 4.7. Let $f_5(z)$ be the following function:

$$f_5(z) = \exp\left(z^2 \log \frac{1}{1-z^4}\right) + \frac{1}{1-2z^6}.$$

An application of the algorithm leads to the computation of the radii of convergence 1 and $2^{-1/6}$ for the two terms. Oracle $\mathcal{O}_1$ then declares the second quantity to be the smaller one. Next we apply the algorithm recursively on $Q(2z^6)$. This needs the direct reduction of $2z^6$ from which we deduce that the singular directions are $\{(2k\pi/6),\ k = 1 \ldots 6\}$.

Note on the role of oracles

We made our first encounter with oracles here. It should be said that we live in a world where the status—transcendent, algebraic, or rational—of constants like

$$\gamma = \int_0^1 \frac{e^{-x}-1}{x}\,dx + \int_1^\infty \frac{e^{-x}}{x}\,dx, \qquad e+\pi, \qquad \zeta(5) = \sum_{n\geq 1} \frac{1}{n^5},$$

is still undecided. Consequently, in view of some the expansions that result from our automatic analysis, it is not too surprising that one should appeal to oracles of sorts.

Fortunately, a reliable oracle of the type $\mathcal{O}_1$ is easily implemented in practice by evaluating the quantities involved numerically, with a high enough precision. It should also be noted that given such an oracle, we can then compute *symbolically* the radius of convergence of functions in $\mathscr{E}$, that is, compute the smallest subexpression of the input function whose root is the smallest positive singularity. The radius of convergence of any function in $\mathscr{E}$ is thus given as a simple root of an elementary equation. We shall henceforth assume that this is the form returned by algorithm Radius.

4.3. Singular growth

All the algorithms we have presented so far work with any function of $\mathscr{E}$. We now isolate a subclass $\mathscr{E}_{\mathrm{AL}}$ of elements of $\mathscr{E}$ for which one can automatically compute the asymptotic expansion of the Taylor coefficients. This class is characterized by the moderate growth of its elements about their singular points.

We actually construct a partition of $\mathscr{E}$ into three disjoint classes,

$$\mathscr{E}_{\mathrm{AL}}, \mathscr{E}_{\mathrm{entire}}, \mathscr{E}_{\exp} \quad \text{such that } \mathscr{E} = \mathscr{E}_{\mathrm{AL}} \cup \mathscr{E}_{\mathrm{entire}} \cup \mathscr{E}_{\exp}. \tag{31}$$

The class $\mathscr{E}_{\mathrm{entire}}$ consists of all entire functions, and it is clearly a decidable subclass of $\mathscr{E}$. The class $\mathscr{E}_{\mathrm{AL}}$ consists of functions with a radius of convergence $\rho < \infty$ and with a so-called *algebraico-logarithmic* (AL) growth,

$$f(z) \sim \frac{1}{(1-z/\rho)^\alpha} \log^k \frac{1}{1-z/\rho}, \qquad z \to \rho^-,$$

where α is real and k is a non-negative integer. These are our main objects of study.

In this subsection we show that one can decide membership to $\mathscr{E}_{\mathrm{AL}}$ thanks to a *gap* property. In the class $\mathscr{E}\setminus\mathscr{E}_{\mathrm{entire}}$ of functions singular at ρ, it is found that functions of the form

$$\exp\left(c \log^2 \frac{1}{1-z/\rho}\right), \qquad c>0,$$

play a special role as threshold functions. The class $\mathscr{E}_{\exp}$ will be characterized by the fact that its elements f grow too fast and are easily recognizable by the property that

$$\log f(z) \geq \log^2 \frac{1}{1-z/\rho} \quad \text{as } z \to \rho^-.$$

Proposition 4.8. *Conditioned upon oracle $\mathcal{O}_1$, membership of a function in $\mathscr{E}_{AL}$ is decidable. A singular expansion of a function in $\mathscr{E}_{AL}$ around its positive dominant singularity is also computable.*

Proof. We define 3 classes $\mathscr{E}_{AL}\{\rho\}$, $\mathscr{E}_{regular}\{\rho\}$, $\mathscr{E}_{exp}\{\rho\}$, that reflect the partition (31) at $z=\rho$. These are respectively, the functions with an algebraic-logarithmic singularity at ρ, the functions regular at ρ, and the functions with an exponential singularity at ρ. Proving that $f\in\mathscr{E}_{AL}$ is equivalent to proving that $f\in\mathscr{E}_{AL}\{\rho\}$ for ρ the dominant singularity of f, a quantity that is assumed to be known from our previous algorithms. Finding the nature of the singularity is done by the following algorithm.

The algorithm essentially composes generalized algebraic-logarithmic expansions. If a fast growing function is detected, the algorithm only returns the proposition "$f\in\mathscr{E}_{exp}\{\rho\}$".

Algorithm Expansion
Input: A function f in $\mathscr{E}$ and a positive real number ρ.
Output: An asymptotic form of f at ρ, or the answer "$f\in\mathscr{E}_{exp}\{\rho\}$".
Comment: Assume $\rho\leq R$, with R the radius of convergence of f. Also assume that ρ is itself the root of an elementary equation.

(1) Compute the radius of convergence R of f by the algorithm Radius.
(2) Appeal to $\mathcal{O}_1$ for deciding whether $R>\rho$ or $R=\rho$.
(3) If $R>\rho$, then return $f(\rho)+f'(\rho)(z-\rho)+O((1-z/\rho)^2)$.
(4) Otherwise, $R=\rho$. Consider cases according to the nature of f.
(a) If f is $Q(g)$ then return

$$\frac{1}{\rho g'(\rho)}\frac{1}{1-z/\rho}+\frac{g''(\rho)}{2g'(\rho)}+O(1-z/\rho).$$

(b) If f is $L(g)$ then if (by $\mathcal{O}_1$) $\rho g'(\rho)\neq 1$ then return

$$\log\frac{1}{1-z/\rho}-\log(\rho g'(\rho))+O(1-z/\rho),$$

else return

$$\log\frac{1}{1-z/\rho}+\frac{\rho g''(\rho)}{2g'(\rho)}(1-z/\rho)+O((1-z/\rho)^2).$$

(c) If $f=g+h$ or $f=g\cdot h$, then apply Expansion to both g and h and add or multiply the results, returning $\mathscr{E}_{exp}\{\rho\}$ if one of the results was $\mathscr{E}_{exp}\{\rho\}$.

(d) If f is $\exp(g)$ then apply Expansion to g, and discuss according to the result: if it is of the form

$$a\log\frac{1}{1-z/\rho}+b+O(1-z/\rho), \tag{32}$$

then return

$$\frac{e^b}{(1-z/\rho)^\alpha}+O\left(\frac{1}{(1-z/\rho)^{\alpha-1}}\right),$$

otherwise return $\mathscr{E}_{\exp}$.

The complete proof of the proposition reduces to checking the correctness of the algorithm; see Salvy's thesis [73] for details. □

Example 4.9. Consider the function f_6 defined as follows:

$$f_6(z)=\exp\left(z\log\frac{1}{1-4z^4}e^{z/(1-z^3)}\right)+e^z\log\frac{1}{1-2z^2}+1+z.$$

Applying algorithm Radius, we find that the radius of convergence is $1/\sqrt{2}$. Then we apply Expansion to $(f_6, 1/\sqrt{2})$. Since f_6 is a sum, the algorithm is called recursively on each of the summands. The algorithm yields

$$\frac{4^{-\alpha}}{(1-z\sqrt{2})^\alpha}+O\left(\frac{1}{(1-z\sqrt{2})^{\alpha-1}}\right),$$

with

$$\alpha=\frac{\sqrt{2}}{2}\exp\frac{4\sqrt{2}+2}{7}.$$

4.4. Asymptotic analysis of coefficients

The singular expansions that we have computed can now be *transferred* to coefficients. The case of a unique dominant singularity is naturally simpler (Theorem 4.10), but information can also be obtained in several periodic cases (Theorem 4.13).

Theorem 4.10. *Let f be an algebraic-logarithmic function of $\mathscr{E}_{\mathrm{AL}}$ with a unique dominant singularity. Then, the coefficients of f have an asymptotic form*

$$f_n=[z^n]f(z)=C\rho^{-n}n^s\log^k n\left(1+O\left(\frac{1}{\log n}\right)\right).$$

There, C, ρ, s are real numbers, k is an integer. Using oracle $\mathcal{O}_1$, all these numbers are computable numerically and expressible in terms of elementary functions and roots of elementary equations.

(Better error estimates are available, but the statements become naturally more complicated; see Salvy's thesis for details [73].)

Proof. The class $\mathscr{E}_{\mathrm{AL}}$ is a source of standardized singular expansions that can now be exploited by means of two theorems. The first theorem, due to Flajolet and

Odlyzko [32], is a variant of the classical Darboux method. For its statement, we need the following notation for Camembert domains,

$$\Delta = \{z, |z| \leq 1+\eta, |\arg(z-1)| \geq \varphi\},$$

for some $\eta > 0$ and $0 < \varphi < \pi/2$.

Lemma 4.11 (Flajolet-Odlyzko [31]). *Assume that $f(z)$ is analytic in $\Delta\backslash\{1\}$, and that as $z \to 1$ in Δ,*

$$f(z) = O\left((1-z)^{-\alpha} \log^{\gamma} \frac{1}{1-z}\right),$$

for some real numbers α, γ. Then the n-th Taylor coefficient of $f(z)$ satisfies

$$[z^n] f(z) = O(n^{\alpha-1} \log^{\gamma} n).$$

Note that the condition of being analytic in a domain larger than the circle of convergence is always fulfilled by our functions which are composed of a finite number of entire and meromorphic functions.

The next theorem is older and was stated by Jungen in 1931 [50].

Lemma 4.12 (Jungen [50]). *Define the coefficients a_n by the expansion*

$$(1-z)^{-s} \log^k \frac{1}{1-z} = \sum_{n=0}^{\infty} a_n z^n,$$

where $k \geq 0$ is an integer and s is an arbitrary complex number. Then the coefficients a_n are given asymptotically by the following formulae:

(1) *If $s \neq 0, -1, -2, \ldots$, then*

$$a_n = \frac{n^{s-1}}{\Gamma(s)} [\log^k n\, \varphi_0(n) + \log^{k-1} n\, \varphi_1(n) + \cdots + \varphi_k(n)],$$

where

$$\varphi_0(n) \sim 1 + c_{01}/n + c_{02}/n^2 + \cdots$$

$$\varphi_1(n) \sim c_{10} + c_{11}/n + c_{12}/n^2 + \cdots$$

$$\vdots$$

(2) *If $s = 0, -1, -2, \ldots$ and $k > 0$, then*

$$a_n = (-1)^s k! \Gamma(1-s) n^{s-1} [\log^{k-1} n\, \varphi_0(n) + \log^{k-2} n\, \varphi_1(n) + \cdots + \varphi_{k-1}(n)]$$

with the φ_i as in (1).

The proof provided by Jungen can be turned into an algorithm for the computation of the c_{ij}, but a more efficient algorithm can be extracted from proofs of similar theorems in [32].

The proof of the theorem is easily completed. Each function satisfying the assumptions has a singular expansion of the form given by Proposition 4.8. Coefficients of the main term can be extracted by Jungen's theorem; the remainder term is itself amenable to Lemma 4.11.

This double transfer completes the proof of the theorem, and the underlying algorithm constitutes the algorithm Transfer referred to in the introduction to this section. □

Many of the examples we have considered so far fall into this class, which permits us to complete the proofs of a few statements made earlier in anticipation of asymptotic methods. Our simple examples here all have explicit singularities that can be found by a reasonable computer algebra system. A first batch deals with meromorphic functions.

Automatic Theorem 12. *The asymptotic number of* 1-2 *compositions of n is*

$$[z^n]\frac{1}{1-z-z^2}=\frac{\phi^{n+1}}{\sqrt{5}}+O\left(\frac{\phi^n}{n}\right) \text{ where } \phi=\frac{1+\sqrt{5}}{2}.$$

The asymptotic number of ordered partitions of n is

$$\left[\frac{z^n}{n!}\right]\frac{1}{2-e^z}=\frac{1}{2\log 2}\frac{n!}{(\log 2)^n}+O\left(\frac{n!}{n(\log 2)^n}\right).$$

The number of derangements D_n satisfies

$$D_n=\left[\frac{z^n}{n!}\right]\frac{e^{-z}}{1-z}=e^{-1}n!+O\left(\frac{n!}{n}\right).$$

Proof. The first two examples (they are also functions f_6 and f_1 of our example list) are direct consequences of our algorithms: we have a rational function and a meromorphic function with an explicit dominant singularity at $\rho=\log 2$. The case of derangements follows that of ordered partitions. □

The second batch of examples is relative to functions with algebraic and logarithmic singularities.

Automatic Theorem 13. *The mean number of cycles in a random permutation of n is*

$$[z^n]\frac{1}{1-z}\log\frac{1}{1-z}=\log n+\gamma+\frac{1}{2n}-\frac{1}{12n^2}+\frac{1}{120n^4}+O\left(\frac{1}{n^6}\right).$$

The asymptotic number of unary-binary expressions of size n is

$$\text{Expression}_n=[z^n]\frac{1-z-\sqrt{1-2z-3z^2}}{2z}=\sqrt{\frac{3}{4\pi n^3}}3^n+O\left(\frac{3^n}{n^{5/2}}\right).$$

The asymptotic number of generalized bracketings of size n is

$$\mathrm{GB}_n = [z^n]\tfrac{1}{4}(1+z-\sqrt{1-6z+z^2}) = \frac{-4+3\sqrt{2}}{4\sqrt{\pi n^3}}(3+\sqrt{2})^n\left(1+O\left(\frac{1}{n}\right)\right).$$

The expected path length in a random plane tree of size n is asymptotically

$$\tfrac{1}{2}\sqrt{\pi}\, n^{3/2}+\tfrac{1}{2}n+O(n^{1/2}).$$

Notice that several examples here are relative to implicitly defined *recursive structures.* As we have seen, the corresponding generating functions are all expressible in closed form. The square-root singularities are amenable to the treatment given here for the class $\mathscr{C}_{\mathrm{AL}}$.

The last batch of examples deals with the more general case of functions whose singularities are defined as roots of various elementary equations.

Automatic Theorem 14. *The following coefficient expansions hold. Let*

$$f_2(z) = \frac{1}{1-\log[1/(1-z-z^2)]},$$

then

$$[z^n]f_2(z) = \frac{e^{-1}}{\frac{5}{2}-\frac{1}{2}\sqrt{5-4e^{-1}}-2e^{-1}}\alpha^{-n}+O\left(\frac{\alpha^{-n}}{n}\right)$$

where $\alpha = \frac{1}{2}(\sqrt{5-4e^{-1}}-1)$. *Let*

$$f_4(z) = \log\frac{1}{1-z\log[1/(1-z^2)]}+\frac{1}{(1-z^3)^5}+\exp(z\,e^z),$$

then

$$[z^n]f_4(z) = \frac{1}{n\rho^n}+O\left(\frac{1}{n^2\rho^n}\right),$$

where ρ *is the smallest positive root of*

$$1 = \rho\log\frac{1}{1-\rho^2}.$$

Let

$$f_7(z) = \exp\left(\log\frac{1}{1-z-z^2}e^{z/(1-z)}\right),$$

then

$$[z^n]f(z) = \left(\frac{\phi}{\sqrt{5}}\right)^{e^\phi}\frac{n^{e^\phi-1}\phi^n}{\Gamma(e^\phi)}+O(n^{e^\phi-2}\phi^n),$$

where ϕ *is the golden ratio.*

Proof. For f_2, the singularity can be made explicit. Note the example of f_4 for which the singularity ρ could not be written in closed form, but this raised no difficulty in the automatic computation. Function f_7 corresponds to a function with a curious singular behaviour at ϕ. □

4.5. Periodicities

The general case of functions in $\mathscr{E}_{AL}$ with several dominant singularities needs treatment. Such functions are called *periodic* (purely periodic functions introduced earlier in Section 4.2 are periodic). A singular expansion must be computed around each dominant singularity, and then translated by the previous theorems into a partial expansion. These partial expansions are then added together, and this yields an expansion for the coefficients.

Difficulties may arise for some functions that do not differ too much from a purely periodic function. In that case, the main asymptotic terms cancel for certain values of the index n, and subdominant terms dictate the asymptotic behaviour of coefficients in this case. There is then an exponential cancellation to be taken into account.

Consider for instance the problem of extracting

$$[z^n]f(z) \text{ where } f(z)=\frac{1}{1-2z^4}+\frac{1}{1-z},$$

from this point of view. We have

$$[z^{4n}]\frac{1}{1-2z^4}=2^n, \qquad [z^n]\frac{1}{1-z}=1.$$

Thus, $f_{4n}=2^n+1$, but for $n\not\equiv 0 \pmod 4$, we have $f_n=1$. When computing f_n, the asymptotic forms deriving from the singularities $2^{-1/4}e^{ik\pi/2}$ $(k=0,1,2,3)$ cancel *exactly* when n is not a multiple of 4. There's the rub!

In fact, by performing only real computations, it is possible to derive an interesting (though not always complete) part of the asymptotic information.

Theorem 4.13. *The coefficients of a function of $\mathscr{E}_{AL}$ with $p\geq 2$ dominant singularities satisfy an asymptotic estimate of the form*

$$f_n=\rho^{-n}n^{s}\log^k n\left[C+\sum_{j=1}^{p-1}C_j\cos\frac{2jn\pi}{p}\right]+O(\rho^{-n}n^{s}\log^{k-1}n),$$

where C, ρ, s, k are as in Theorem 4.10, *and $0\leq C_j\leq C$. Using oracle $\mathcal{O}_2$, the constants C_j can be computed.*

Oracle $\mathcal{O}_2$ is defined as follows.

Definition 4.14. Given a function $f\in\mathscr{E}$ and a dominant (complex) singularity $\sigma=\rho e^{2i\pi p/q}$ of another function of $\mathscr{E}$ such that f is regular at σ, oracle $\mathcal{O}_2$ outputs one

of the assertions

$$f(\sigma)=0, \qquad f(\sigma)\neq 0.$$

Under several circumstances the expansion we obtain from this theorem is a bona fide asymptotic equivalent. Difficulties may arise if the fluctuating sum of cosines vanishes, in which case we only get a rather weak $O(.)$ estimate for f_n. At least, what stays is a dominant asymptotic regime that describes the behaviour of a "dense" fraction (i.e., a non-zero proportion) of the f_n.

Proof. The basic idea consists of computing expansions related to singular terms of the same form as before,

$$\left(1-\frac{z}{\sigma}\right)^{-\alpha}\log^{k}\frac{1}{1-z/\sigma},$$

except that now σ may be complex, $\sigma=\rho\,e^{i\theta}$, with ρ the radius of convergence of the series under consideration and θ a real angle.

The translation to coefficients is effected like before, and each dominant singularity may contribute. We should simply add the corresponding contributions. The problem then lies with the precise determination of the constants C_i. Although they are computed by the same type of expansions as in the real case, this calculation presents an additional difficulty: we have to decide whether a function in $\mathscr{E}$ which is regular at a complex point (the singularity under study) is zero there or not. This task is accomplished by oracle $\mathcal{O}_2$. When examining the behaviour of a function f at σ, we modify step 3 of algorithm Expansion in the following way:

(a) Check whether f is a polynomial or not (this is decidable in $\mathscr{E}$).

(b) If it is not, compute values of $f^{(k)}(\sigma)$, $k\geq 0$ until $\mathcal{O}_2$ finds two of them, $f^{(k_1)}$, $f^{(k_2)}$, to be non-zero. Return

$$f^{(k_1)}(\sigma)(z-\sigma)^{k_1}/k_1!+f^{(k_2)}(\sigma)(z-\sigma)^{k_2}/k_2!+O((z-\sigma)^{k_2+1}).$$

(c) If it is, do the same operations but stop when $k>\deg(f)$. Return the expansion so obtained.

The rest of algorithm Expansion works unchanged, mainly because, in steps 4a and 4b, g is necessarily real on the ray from 0 to σ. □

A few of the functions we have met in our examples present several singularities on their circle of convergence. The first case is easy, since a single step of reduction suffices to analyze the coefficients of the functions involved.

Automatic Theorem 15. *The expected number of cyclic points in a random binary functional graph of size n is for* $n\equiv 0 \pmod 2$,

$$\sqrt{\frac{\pi}{2}}\,n^{1/2}-1+O(n^{-1/2}).$$

(The functions involved in this analysis are purely periodic, being simply functions of z^2, see Automatic Theorem 11.)

The next batch of examples illustrates more intricate situations, and in one case only partial coefficient information is available from the algorithms we have just discussed.

Automatic Theorem 16. *The following asymptotic coefficient expansions hold.*

$$f_3(z)=\frac{1}{\sqrt{1-z}}\frac{1}{1-z^3} \Rightarrow [z^n]f_3(z)=\frac{2}{3}\sqrt{\frac{n}{\pi}}+O(1).$$

$$f_5(z)=\exp\left(z^2\log\frac{1}{1-z^4}\right)+\frac{1}{1-2z^6} \Rightarrow [z^{6n}]f_5(z)=2^n+O\left(\frac{2^n}{n}\right)$$

$$f_6(z)=\exp\left(z\log\frac{1}{1-4z^4}e^{z/(1-z^3)}\right)+e^z\log\frac{1}{1-2z^2}+1+z,$$

$$\Rightarrow [z^n]f_6(z)=\frac{\sqrt{2}^n n^{\alpha-1}}{4^\alpha\Gamma(\alpha)}+O(n^{\alpha-2}\sqrt{2}^n),$$

with

$$\alpha=\frac{\sqrt{2}}{2}\exp\frac{4\sqrt{2}+2}{7}.$$

Proof. In the case of f_3, the complex singularities are of a smaller order than the real one. This is not the case with f_5 for which we only get an expansion for indices n that are multiples of 6. More complete estimates can be found for products like f_6, where we get all coefficients. □

Even though in case of periodicities the raw version of the algorithms we have described is not guaranteed to produce an asymptotic expansion for all coefficients, in practice an expansion can often be obtained by computing singular expansions with more terms. We do not attempt to develop the theory in this case as it becomes naturally rather intricate.

5. Extensions

We discuss here some of the possible extensions of our approach. Clearly, theoretical advances can be used to great advantage in extending the functionality of an automatic analysis system.

We saw (briefly) in Section 1 that there are three components in our automatic analysis system. The two major ones correspond to the computation of generating functions (the algebraic analyzer) based on the algebraic enumeration techniques of Section 3 and to the asymptotic analysis (the analytic analyzer) based on the function-theoretic techniques of Section 4. The interface is ensured by the solver. Our discussion will follow this template.

5.1. Algebraic counting

We have seen how to analyze four major classes in terms of automatically determined generating functions. Each new combinatorial construction or programme control structure that admits a translation into GFs will enable us to solve an enlarged class of problems.

Minimum rooting

Many algorithms deal with ordered structures. A particularly interesting construction that fixes the localization of the smallest label in a labelled structure (min-rooting) has been formalized by Greene in his thesis [40] under the name of *box operator.* The equation

$$\mathscr{A} = \mathscr{B}^{\square} * \mathscr{C} \tag{33}$$

means that $\mathscr{A}$ is the usual partitional product of $\mathscr{B}$ and $\mathscr{C}$, with the condition that the smallest label lies in the $\mathscr{B}$-component. Greene has proved that labelled context-free grammars augmented with the box construction translate over generating functions through integro-differential operators. For instance, in the case of (33), we have the recurrence,

$$A_n = \sum_{j=1}^{n} \binom{n-1}{j-1} B_j C_{n-j},$$

where the modified binomial coefficient takes care of the fact that only $n-1$ labels need to be distributed between $\mathscr{B}$ and $\mathscr{C}$. In terms of EGFs, this means

$$A(z) = \int_0^z \left(\frac{\mathrm{d}}{\mathrm{d}t} B(t)\right) C(t)\, \mathrm{d}t. \tag{34}$$

It can be shown that an interesting set of programmes on structures defined with the box operator are admissible too. They then translate into differential equations for the complexity descriptors. We will not give the complete rules here; they will appear in [90]. Let us just cite an example, that of heap-ordered trees.[9]

Automatic Theorem 17. *The average internal pathlength in a heap-ordered tree of size n is asymptotically*

$$2n \log n + (2\gamma - 3)n + 2 \log n + O(1).$$

Proof. Heap-ordered trees admit the specification,

```
type Heap = leaf | min(key) Heap Heap;
     key = Latom(1);
     leaf = Latom(0);
```

[9] A heap-ordered tree is a labelled binary tree with labels that are in increasing order along any branch starting from the root. Such trees have strong relations to binary search trees and quicksort. We refer to Vuillemin's article [82] for a discussion of combinatorial aspects of these trees that are known there as "tournament" trees.

The programme for analyzing internal pathlength follows closely the one used for general plane trees.

```
procedure PathLength (h : Heap );
begin
  size(h);
  case h of
    leaf : zero;
    (k,h1,h2) : begin PathLength(h1); PathLength(h2) end;
  end;
end;

procedure size (h : Heap );
begin
  case h of
    leaf : zero;
    (k,h1,h2) : begin one; size(h1); size(h2) end;
  end;
end;

measure one : 1;
        zero : 0;
```

The rule (33-34) gives us an equation for Heap(z), namely,

$$\mathrm{Heap}(z) = 1 + \int_0^z \left(\frac{\mathrm{d}}{\mathrm{d}t}\, t\right)(\mathrm{Heap}(t))^2\, \mathrm{d}t,$$

an equation that leads to a non-linear differential equation with variables that separate,

$$\frac{\mathrm{d}}{\mathrm{d}z}\, Y(z) = Y^2(z), \qquad Y(0) = 1.$$

Such problems are well within the capabilities of computer algebra systems, and one finds, as expected,

$$\mathrm{Heap}(z) = \frac{1}{1-z}.$$

The rules given in [90] allow us further to compute the complexity descriptor of the procedure PathLength, and the literal form produced by $\Lambda\gamma\Omega$ is in this case

$$\tau\mathrm{PathLength}(z) = -\frac{z}{z^2-2z+1} - \frac{2\log(1-z)}{z^2-2z+1}.$$

The automatic theorem follows from these functions and from theorems of Section 4. □

This analysis is of special interest as it relates to the analysis of binary search trees and of the quicksort algorithm.

Boolean functions

The programme constructions of Section 3 operate with "pure" procedures in which no result is ever passed and reused by another procedure. This corresponds to our general and informal notion of pure traversal procedures. The possibilities for extensions in this area are of course limited by undecidability considerations. However, a nice class of functions returning boolean values can also be integrated into the system. An example of this is a programme that checks the occurrence of certain symbols in binary trees.

```
type zero,X,g = atom(1);
     T = zero | X | product(g,T,T);
function Occurs(t : T) : boolean;
begin
  Visit;
  case t of
    zero : false;
    X : true;
    (g,u,v) : if Occurs(u) then true else Occurs(v);
  end;
end;
measure Visit : 1;
```

This determines whether an expression contains the variable X or not; the cost of `Occurs(t)` is the number of nodes visited. This programme is in none of the four classes defined earlier (see Theorem 3.13), because in the statement "`if Occurs(u) then true else Occurs(v)`" we (recursively) use the result of another computation.

For a boolean function f, several well defined rules allow us to compute two type specifications of data items for which f returns *true* and *false*. Once these type specifications are known, the schemes "`if f(x)=true then . . .`" and "`if f(x)=false then . . .`" become admissible.

Automatic Theorem 18. *Consider expressions built with the symbol* 0, *a variable* X *and a binary operator* g. *The average number of nodes of a random expression of size* n *that are visited in the course of a preorder traversal before finding* X *is*

$$\frac{[z^n]-\dfrac{\sqrt{1-8z^2}-1}{z\sqrt{1-8z^2}+z\sqrt{1-4z^2}}}{[z^n]-\dfrac{\sqrt{1-8z^2}-1}{2z}}.$$

Asymptotically, this quantity is, for n *odd,*

$$4+\sqrt{2}+O(1/\sqrt{n}).$$

Proof. The number of nodes visited is the number of times the function `Visit` is called in the above programme, whence the cost of this programme. The rules given in this paper enable to compute the GF for expressions

$$T(z) = -\frac{\sqrt{1-8z^2}-1}{2z}$$

and the rules given in [91] enable us to compute the complexity descriptor which is found to be

$$\tau\text{Occurs}(z) = -\frac{\sqrt{1-8z^2}-1}{z\sqrt{1-8z^2}+z\sqrt{1-4z^2}}.$$

The asymptotic result follows from direct singularity analysis. □

Such a scheme has been introduced into the $\Lambda\gamma^{\Omega}$ system and it has proved useful in an average-case analysis of several unification algorithms [2].

5.2. *Implicit and explicit generating functions (the Solver)*

The algebraic analysis produces *functional equations*, while the asymptotic analysis techniques that we have used so far require an *explicit form* for generating functions and complexity descriptors.

At the interface between these two components of the analysis process[10] there should (ideally) lie a well defined model of algebraic manipulation. In order not to obscure the picture, we have been discreet so far on this subject. A few explanations mixing theoretical considerations as well as implementation problems will now be offered.

Clearly, some amount of algebraic manipulation is needed, at least because of machine-man interaction. For instance, it may be desirable to incorporate simplification rules such that

$$\exp\left(\log\frac{1}{1-z}\right) \Rightarrow \frac{1}{1-z}.$$

This simplification occurs in the analysis of the cycle decomposition of permutations and is the one that enables us to *conclude* that the EGF of permutations is $1/(1-z)$, i.e., the number of permutations of n is $n!$.

On another register, usual simplifications like

$$X\times 0 \Rightarrow 0,\ X\times 1 \Rightarrow X,\ X+Y+X \Rightarrow 2X+Y,\ \text{RootOf}(x^2-4, x>0) \Rightarrow 2,\ldots,$$

are certainly a necessity. The status of simplification rules of the form

$$\log(X\cdot Y) \Rightarrow \log X+\log Y, \qquad \sqrt{X\cdot Y} \Rightarrow \sqrt{X}\cdot\sqrt{Y},$$

is much more debatable and their usefulness depends upon context.

[10] In the $\Lambda\gamma^{\Omega}$ system, this interface is the function of the Solver module.

In this paper, at theory level, we have generally assumed the common rules of elementary algebra when manipulating generating functions. The designer of an automatic analyzer (like the $\Lambda\Upsilon\Omega$ system) should in principle take full control over the simplification rules that are employed. However, for obvious efficiency reasons, it is usually not possible to enforce such a vigorous policy. So, the algebraic capabilities of the $\Lambda\Upsilon\Omega$ system rely on those of the host computer algebra language, namely the Maple system. As is natural, this occasionally creates conflicts between what is needed of a general purpose computer algebra system and the stricter simplification discipline that a system like $\Lambda\Upsilon\Omega$ requires for its more limited universe of special functions. Examples of such problems are well-known to designers, e.g., the rules

$$(R_0)\ \sqrt{(1-z)^2} \Rightarrow (1-z) \quad \text{and} \quad (R_1)\ \sqrt{(1-z)^2} \Rightarrow \sqrt{(z-1)^2} \Rightarrow (z-1),$$

though being each reasonable under certain conditions, may lead to inconsistent results. (This is in no way meant as a criticism of the Maple system without which the $\Lambda\Upsilon\Omega$ enterprise would not have existed. Such problems are bound to occur with any system currently in existence [61].)

In the sequel, we assume in our discussion that we have available an ideal engine for algebra manipulations.

The two issues to be discussed are: (i) simplication and resolution of equations; (ii) the universes of special functions.

Simplification and resolution of equations

(1) In the universe of purely iterative labelled structures, an equational definition of structures leads to a chain of equations that can be solved by direct substitution. For instance this remark is at the origin of the result that the class of associated GFs is the elementary class $\mathscr{E}$ defined as the closure of 1, z by operators of Ω_{labelled}.

(2) For programmes over labelled recursive structures, the complexity descriptors are plainly given by linear equations over GFs. In that case, explicit solutions are derived automatically from the counting GFs assuming only that a *linear equation solver* is available. In a way, the most difficult part[11] of an analysis is the one relative to counting GFs.

(3) For recursive structures in the labelled universe (and simpler recursive unlabelled structures that do not involve sets or cycles), the GFs appear as fixed point equations over the class of elementary functions. Such equations are in general highly non-linear, but it is possible to trap interesting subclasses.

For instance, we may consider the class of *quadratic structures* in which all GFs are resolved by: (i) substitution; (ii) linear equations; (iii) quadratic equations. There are obvious examples of quadratic structures, the simplest being the pure

[11] This observation reflects the fact that *time* is an *additive* complexity measure that is in a loosely defined sense "linear", and resembles a "derivation". Our approach cannot be extended to space complexity measures that replace sums by *maximum* operators.

binary trees

```
type BB = BB o BB | X;
X = atom(1); o = atom(0);
```

or the unary-binary expression types. Other cases, like the binary functional graphs or the generalized bracketings with OGF

$$\mathrm{GB}(z) = z + \frac{\mathrm{GB}^2(z)}{1-\mathrm{GB}(z)},$$

illustrate the fact that the precise notion of a quadratic structure is relative to a given model[12] of algebraic simplification.

(4) Our understanding of the algebra of Pólya operators Φ (for data types) and Ψ (for programmes) is not too advanced. We only know that certain identities exist, for instance,

$$\Phi_{M^2}(f(z)) = \Phi_S(f(z))\Phi_M(f(z^2)),$$

which corresponds to $(1-z)^{-1} = (1+z)(1-z^2)^{-1}$. This does not seem to be a major drawback however since most of the work in this case should be rejected to the asymptotic analyzer, using techniques that we detail below.

Function universes

The remarks above show that a complete discussion of automatic analysis must include a precise discussion of simplification issues for the class of functions used. This in turn is related to a notion of which *special functions* and corresponding properties are regarded as *known*.

As an example, the family of Cayley trees defined by

```
Cayley = o set(Cayley); o = Latom(1);
```

corresponds to the implicit equation,

$$\mathrm{Cayley}(z) = z\exp(\mathrm{Cayley}(z)).$$

Though this equation could be regarded as not elementarily solvable, the Maple session

```
> solve(Cayley(z)=z*exp(Cayley(z)),Cayley(z));
                                        - W(- z)
```

expresses it in terms of a special transcendental[13] $W(z)$ (the root of $We^W = z$) which is regarded as known by the system. We could thus define *W-structures*, in the same way as we have defined quadratic structures. However, the proliferation of such definitions is best avoided. Probably a more general approach based on an extensive use of standardized implicit functions reusable by the asymptotic analyzer is to be preferred.

[12] Throughout the paper, in our automatic theorems, we have operated with the naïve notion induced by the capabilities of the Maple "solve" routine. Then, for us, GB is a quadratic structure.

[13] This function was considered by Eisenstein and Cayley, amongst others.

Another example is provided by the family of ternary trees. In that case, the specification is

```
type Ternary = o Ternary Ternary Ternary | X;
     X = atom(1); o = atom(0);
```

The corresponding solution is known,

$$\text{Ternary}(z)=\left(-\frac{1}{2}+\frac{\sqrt{-4+27z^2}}{6\sqrt{3}}\right)^{1/3}+\left(-\frac{1}{2}-\frac{\sqrt{-4+27z^2}}{6\sqrt{3}}\right)^{1/3},$$

meaning that ternary trees belong to the class of "*radical*" *structures*. However, the asymptotic analysis of the GF Ternary(z), though feasible from the explicit form, is best carried out by subjecting anonymously Ternary(z) to a general asymptotic treatment of *algebraic functions*.

Finally, we saw in the analysis of heap-ordered trees that certain constructions relating to order constraints introduce integro-differential operators. (The capability of Maple's "dsolve" differential equation solver was used on that occasion.) In this context, an interesting class of functions is that of combinatorial *holonomic systems* of Zeilberger [88], which, in the univariate case, reduces to the class of D-finite functions described by Stanley [77].

In other words, we could also regard as known the solution $Y(z)$ of any equation

$$\sum_{j=0}^{d} Q_j(z)\frac{\mathrm{d}^j}{\mathrm{d}z^j}Y(z)=0,$$

where the Q_j are rational functions, and proper initial conditions completely determine $Y(z)$. This class has rich closure properties. To a large extent, the corresponding asymptotic problems on coefficients are solvable; this results from either Birkhoff's theory of difference equations [85], or from the singularity analysis techniques that we have utilized in Section 4.

Differential operators will not be discussed further here.

5.3. Analytic schemes

In Section 4, we have seen how to analyze a particular subclass of problems arising from labelled iterative functions. In that case, a full characterization of the possible asymptotic behaviours was attained. Our knowledge of the other types is not so systematic. However, several analytic techniques from earlier works can be put to work for us. In this subsection, we propose to summarize the main ideas that extend the automatic approach to a much larger class of problems.

Labelled iterative structures: In order to complete the classification of these structures, we need to analyze elementary functions with "exponential" growth at their dominant singularities, $\mathscr{E}_{\exp}$, and the entire functions $\mathscr{E}_{\text{entire}}$. Saddle point integrals are the major tool for this range of problems.

Labelled recursive structures: The EGFs are then defined implicitly by elementary equations. This vastly generalizes the situation of context-free grammars. Singularity analysis techniques are known to be applicable to several interesting subcases, and they prove to be the essential tool in this range of problems.

Unlabelled structures: The set and cycle constructions lead to Pólya operators that have complicated explicit forms. However, a simple technique that goes back to Pólya and that has been used extensively in analyzing trees and graphs makes singularity analysis applicable to a wide range of problems in this class.

Labelled iterative structures and saddle point analysis

Still starting from Cauchy's formula, the method which is used for functions of faster growth is the *saddle-point method.* It attempts to find a suitable path of integration through a remarkable point called the *saddle point*. The Cauchy integral is concentrated about this point, and it is possible to obtain precise asymptotic information by neglecting the other parts of the contour (see e.g., [25, 63, 86]). The problem with this method is that it is difficult in the most general setting to prove the validity of neglecting these other parts of the path.

In 1956, Hayman [45] delimited a class of functions for which one can compute systematically an asymptotic form of coefficients by a saddle point method. This class of so-called *H-admissible functions* also enjoys nice closure properties which make it a useful tool for *automatic* computations. There are two main theorems in Hayman's theory.

Theorem 5.1 (Hayman [45]). *If* $f = \sum f_n z^n$ *is H-admissible, then as* $n \to \infty$,

$$f_n \sim \frac{f(r)}{r^n \sqrt{2\pi b(r)}},$$

where $r = r(n)$ *is the smallest positive root of* $rf'(r)/f(r) = n$, *and* $b(r) = r\,\mathrm{d}(rf'(r))/\mathrm{d}r$.

This first theorem is typical of the form of estimates that one expects when using saddle-point methods. The second theorem provides closure properties that are important for our purposes.

Theorem 5.2 (Hayman [45]). *Properties of H-admissible functions*:

(1) *Closure property*: *If* f *and* g *are H-admissible,* P *is a polynomial with real coefficients and positive leading coefficient, then* $\exp(f)$, $f+g$, $f+P$, $P(f)$, $P \cdot f$ *are H-admissible.*

(2) *If* P *is an aperiodic polynomial with positive coefficients, then* $\exp(P)$ *is H-admissible.*

(3) *Let* α, β_1 *be positive real numbers and* β_2, β_3 *real numbers, then*

$$f(z) = \exp\left[\beta_1 (1-z)^{-\alpha} \left(\frac{1}{z}\log\frac{1}{1-z}\right)^{\beta_2} \left(\frac{2}{z}\log\left(\frac{1}{z}\log\frac{1}{1-z}\right)\right)^{\beta_3}\right]$$

is H-admissible.

We say here that $P(z)$ is aperiodic when P is not a function of z^p for any $p>1$. The first part of this theorem reduces the test for H-admissibility to simpler tests, the second part provides the basis for most of the entire H-admissible functions, and the last part deals with functions with singularities at a finite distance.

The simplest example of H-admissible function is $\exp(z)$. From it we get Stirling's formula:

$$\frac{1}{n!}\sim\frac{1}{\sqrt{2\pi n}}\left(\frac{e}{n}\right)^n.$$

Another example is provided by Bell numbers.

Automatic Theorem 19. *The number of partitions of size n satisfies*

$$P_n = n![z^n]\exp(e^z-1)\sim n!\frac{e^{e^r-1}}{r^{n+1}\sqrt{2\pi\exp(r)}},$$

where $r=r(n)$ *is the positive root of* $r\exp(r)=n$,

$$r(n)=\log n-\log\log n+\frac{\log\log n}{\log n}+O\left(\frac{\log\log n}{\log^2 n}\right).$$

The next example is typical of what can happen at a finite distance. It deals with so-called "Laguerre configurations". (The name derives from the resemblance of the GF of Laguerre configurations with the GF of Laguerre polynomials.) A Laguerre configuration is a permutation in which each cycle carries exactly one mark. Since we can "open" cycles at their mark, we can define a Laguerre configuration by the specification:

```
type Laguerre = set(OpenCycle);
     OpenCycle = sequence(Node, card>=1);
     Node=Latom(1);
```

From the specification, we obtain the EGF of Laguerre configurations as $\exp(z/(1-z))$.

Automatic Theorem 20. *The number of Laguerre configurations of size n is asymptotically*

$$n![z^n]\exp\frac{z}{1-z}\sim n!\frac{e^{2\sqrt{n}}}{2\sqrt{\pi}\,e n^{3/4}}.$$

Proof. Verify automatically H-admissibility from Theorem 5.2, then insert the saddle point formula of Theorem 5.1. □

An unfortunate drawback of Hayman's method is that it provides only the main term of the expansion of coefficients. Another class of functions was subsequently introduced by Harris and Schoenfeld [44]. By imposing more stringent conditions on the functions in their class, called the class of HS-admissible functions, they were able to derive a full asymptotic expansion of function coefficients. As such, the HS-admissible functions did not lend themselves to a direct implementation until Odlyzko and Richmond [62] noted the following property: if f is H-admissible, $\exp(f)$ is HS-admissible. It then becomes feasible to *automatically* derive a full asymptotic expansion of Bell numbers, for example.

Two gaps are still to be filled. First neither the method of Hayman nor that of Harris and Schoenfeld can produce a full asymptotic expansion for functions of "fast but moderate" growth. Thus one cannot find the other terms in Stirling's formula by their method. This problem has been partially resolved by Wyman [87], but the corresponding class cannot be implemented easily. Also, functions of an even slower growth do not fit in any known class yet; an instance is the slowly growing function

$$\exp\left(z \log^2 \frac{1}{1-z}\right).$$

As a last remark, let us note that it is far from easy to manipulate automatically expansions in such general scales. A good theoretical framework for this kind of work lies in Hardy's tract on "orders of infinity" [43] and in their generalization by Hardy fields [13]. More about this will be said elsewhere [73].

Labelled recursive structures and implicit functions

In a long series of papers, Meir and Moon (see, e.g., [59]), have considered so-called *simple* families of trees (Meir and Moon say "simply generated"). Essentially, these are classes of recursive tree structures, either labelled or unlabelled, whose generating function is defined by an equation

$$f(z) = z\phi(f(z)). \tag{35}$$

(The various tree examples that we have considered so far are closely related to this notion.)

The interest of this class for us is to provide a protypical treatment of recursive structures. We may as well assume without great loss of generality that $\phi(y)$ is a polynomial in y with positive coefficients, in which case (35) implicitly defines $f(z)$ as an algebraic function of its argument.

In outline, the analysis of the coefficients of f proceeds as follows. Let (z_0, y_0) be a point on the algebraic curve defined by (35), so that $y_0 = z_0\phi(y_0)$, or equivalently $y_0 = f(z_0)$. Let $P(z, y) = y - z\phi(y)$. By expanding the equation $P(z, y) = 0$, we obtain

$$(z - z_0)\frac{\partial}{\partial z} P(z_0, y_0) + (y - y_0)\frac{\partial}{\partial y} P(z_0, y_0) \sim 0,$$

locally. Thus, around an ordinary point, we have a linear dependence between $Z = z - z_0$ and $Y = y - y_0$, provided that the partial derivatives are non-zero. A close examination reveals that the dependence is actually analytic.

The analytic dependence breaks down when the partial derivative with respect to y vanishes. In that case, pushing to the next order in y, we find a relation

$$(z - z_0)\frac{\partial}{\partial z}P(z_0, y_0) + \frac{1}{2}(y - y_0)^2 \frac{\partial^2}{\partial y^2}P(z_0, y_0) \sim 0. \tag{36}$$

Thus the function $f(z)$ admits a branch point at z_0; its value there is y_0, and its singularity is of the square-root type, as seen from the approximate solution of (36),

$$f(z) \sim y_0 \pm \lambda\sqrt{z_0 - z}, \qquad \lambda^2 = 2\frac{\dfrac{\partial}{\partial z}P(z_0, y_0)}{\dfrac{\partial^2}{\partial y^2}P(z_0, y_0)}.$$

Thus, for a singularity, z_0 and y_0 are algebraic numbers determined by a system of two equations. Details can be worked out for the particular equation (35). With τ the positive root of $\tau\phi'(\tau) = \phi(\tau)$, the dominant singularity of f is $\rho = \tau/\phi(\tau)$, and the square root growth yields a coefficient of the form $\approx \rho^{-n} n^{-3/2}$.

Theorem 5.3 (Meir and Moon [59]). *The coefficient of z^n inside the implicitly defined function*

$$y(z) = z\phi(y(z))$$

has the asymptotic form

$$f_n \equiv [z^n]f(z) \sim \delta\rho^{-n}n^{-3/2}, \quad \delta = \sqrt{\frac{\phi(\tau)}{2\pi\phi''(\tau)}},$$

where $\tau\phi'(\tau) = \phi(\tau)$.

In summary, this approach consists of looking at places where the implicit function theorem fails to provide an analytic solution. This defines a collection of elementary equations amongst which the singularities of implicit functions lie. By expanding further, we obtain non-linear dependencies resulting in branch points (through inversion). These algebraic branch points cause coefficients to be composed *asymptotically* of algebraic elements of the form $\rho^{-n}n^{p/q}$ with p, q integers.

The "implicit function method" applies *par excellence* to the coefficients of arbitrary algebraic functions, see [30] for enumerative applications. It is applicable to a large class of transcendentals, such as the Cayley function. It must constitute the method of choice when attempting a complete asymptotic classification of labelled recursive structures via appropriate multidimensional generalizations.

Unlabelled structures and Pólya operators

We start the discussion by two examples, integer partitions and multisets of words (which we call here "languages").

```
type Partition = multiset(Integer);
     Integer = sequence(One, card>0);
     One = atom(1);

     Language = multiset(Word);
     Word = sequence(Letter, card>0);
     Letter = a | b; a, b = atom(1);
```

The corresponding OGFs $P(z)$, $L(z)$ are

$$P(z)=\prod_{n=1}^{\infty}(1-z^n)^{-1}=\exp(I(z)+\tfrac{1}{2}I(z^2)+\tfrac{1}{3}I(z^3)+\cdots)$$

$$L(z)=\prod_{n=1}^{\infty}(1-z^n)^{-2^n}=\exp(W(z)+\tfrac{1}{2}W(z^2)+\tfrac{1}{3}W(z^3)+\cdots),$$

with

$$I(z)=\frac{z}{1-z}\quad\text{and}\quad W(z)=\frac{2z}{1-2z}.$$

It turns out that the seemingly innocuous difference between $I(z)$ and $W(z)$ has implications for the analysis.

The asymptotic theory of $P_n=[z^n]P(z)$ is a classical chapter (originating with Hardy and Ramanujan) of additive analytic number theory. In full generality, it involves a mixture of saddle-point analysis, modular transformations, Dedekind sums, and Ford circles!

One main point is that $P(z)$ has a natural boundary at $|z|=1$. The end result is given by Rademacher's form of the Hardy-Ramanujan theorem [3, p. 69].

Theorem 5.4 (Hardy-Ramanujan-Rademacher). *The number of integer partitions of n is*

$$P_n=\frac{1}{\pi\sqrt{2}}\sum_{k=1}^{\infty}A_k(n)k^{1/2}\left[\frac{\mathrm{d}}{\mathrm{d}x}\frac{\sinh(\pi/k)(\frac{2}{3}(x-1/24))^{1/2}}{(x-1/24)^{1/2}}\right]_{x=n}\tag{37}$$

where

$$A_k(n)=\sum_{\substack{h \bmod k\\(h,k)=1}}\omega_{h,k}\,\mathrm{e}^{-2\pi\mathrm{i}nh/k},$$

with $\omega_{h,k}$ a certain 24th root of unity.

The full analysis of P_n, as suggested by the statement of the theorem, is difficult. In contrast, the asymptotic analysis of $L(z)$ needs only a few lines.

Theorem 5.5. *The number of "languages" of size n is asymptotically*

$$L_n \sim \frac{e^{\delta}}{2\sqrt{\pi e}\, n^{3/4}} e^{2\sqrt{n}} 2^n \quad \text{where } \delta = \sum_{k=1}^{\infty} \frac{1}{k+1} \frac{1}{2^k - 1}.$$

(Such a theorem is in principle well within the capabilities of an automatic analyzer; it should really be an "automatic theorem", but in the current implementation of ΛΥΩ, Pólya operators are not yet taken into account by the asymptotic modules.)

Proof. We observe that $W(z)$ is singular at $\rho = \frac{1}{2}$, and it has a simple pole there. The crucial point is that we have $\rho < 1$, so that when z is in the vicinity of ρ, the arguments of $W(z^2)$, $W(z^3)$, ..., are near $\rho^2, \rho^3, \ldots$, that is to say well *within* the disk $|z| < \rho$. Simple bounds show further that the series

$$\Lambda(z) = \tfrac{1}{2}W(z^2) + \tfrac{1}{3}W(z^3) + \cdots$$

is *analytic* at $z = \rho = \frac{1}{2}$.

Thus, from an asymptotic standpoint, our problem is reduced to analyzing a simple function, namely

$$\exp\left(\frac{2z}{1-2z}\right) e^{\Lambda(z)},$$

where $\Lambda(z)$ is analytic in $|z| < \rho^{1/2} = 2^{-1/2}$. The saddle-point formula

$$[z^n] \exp\left(\frac{z}{1-z}\right) \sim \frac{e^{2\sqrt{n}}}{2\sqrt{\pi e}\, n^{3/4}},$$

yields the result: we need to change z to $2z$ which multiplies this form by 2^n. The influence of the Pólya operator is miraculously (?!) limited to the simple factor $e^{\Lambda(1/2)}$. □

The structures which, like partitions, have a radius of convergence equal to 1 are (decidably) isolated within the class of unlabelled structures. These require special treatment. (Observe however that saddle-point techniques readily provide an asymptotic equivalent of the number of partitions.)

Apart from this small fragment, unlabelled iterative structures can only lead to *isolated* singularities. The composition rules for singularities are easily extracted from the forms

$$\Phi_C(f) = \log(1 - f(z))^{-1} + f_1(z),$$

$$\Phi_S(f) = \exp(f(z)) \cdot f_2(z),$$

$$\Phi_M(f) = \exp(f(z)) \cdot f_3(z),$$

where f_1, f_2, f_3 are analytic in larger areas. In other words, with respect to singularity analysis and saddle point, the remainders play the role of additive or multiplicative modifiers that do not affect the nature of singularities, as we have seen in the case

of "languages". The types of singularities (either algebraic-logarithmic or exponential) remain of the same form as in the labelled case. The asymptotic shape of coefficients thus remain decidable, though special constants (like δ above) are introduced.

We shall not attempt any discussion of unlabelled recursive structures. The techniques there mix what we have just seen concerning unlabelled iterative structures together with ideas stemming from the analysis of singularities for implicitly defined generating functions. The reader should turn to the literature on graphical enumerations [41], and especially to a paper by Harary et al. where a subclass of asymptotic problems on graph trees is shown to be decidable [42].

6. Conclusions

A coherent class of elementary combinatorial problems can only lead to designated "special" asymptotic forms.

We have mentioned in the introduction that properties definable by regular languages (equivalently finite automata) and context-free languages all lead to asymptotic expressions involving "rational" or "algebraic" asymptotic elements of the form

$$P(n)\omega^n \quad \text{and} \quad n^{r/s}\omega^n,$$

for algebraic numbers ω.

As a particular case, the *asymptotic density* of an unambiguous context-free language can only be an algebraic number, which constitutes Berstel's density theorem [8]. Generalized densities for context-free languages exist and they are built from the algebraic elements described above. This means that any elementary counting property of context-free languages can only involve exponentials and rational powers of n but no logarithm. This observation is in agreement with standard probability theory: for instance, the probability that a coin-tossing sequence of length $2n$ is well-balanced (has n tosses of each type) is asymptotic to $1/\sqrt{\pi n}$. (Negative results also derive from this: for instance, square-free numbers and prime numbers in binary representations cannot form context-free languages since the corresponding arithmetic densities are $\sim 6/\pi^2$ and $\sim 1/\log n$ respectively.)

If we look back at the four combinatorial examples that we discussed at the beginning of the introduction, we observe that they all deal with elementary combinatorial objects, namely cycles, permutations, heap-ordered trees, expression trees, and elementary properties like number of components or cost of recursive transformations.

The *symbols* used in the results are all related to classical functions of analysis, and we found "elements" like

$$\exp(-1),\ \sqrt{3},\ \sqrt{\pi},\ \log n,\ \sqrt{n},$$

intervening in the asymptotic solutions.

The discussion in Sections 3–5 should explain why this is so. Our purpose is now to put the results and the methods of earlier sections in a broader perspective. This is achieved by means of structure theorems.

Structure theorems

In this discussion, it proves convenient to use the same naming convention for classes of structures and classes of equations satisfied by their generating functions.

In this way, we speak of rational structures for structures defined rationally, i.e., structures definable by finite automata and regular languages. In the same perspective, algebraic structures correspond to context-free languages. (These two conventions are in agreement with the naming conventions of the "French School".)

In Section 4, we have developed a part of the theory of elementary *labelled iterative* (LI) structures, that of *algebraic-logarithmic* structures defined as those elementary LI-structures whose GFs lie in $\mathscr{E}_{\mathrm{AL}}$. In Section 5.3, we have provided indications on the analytic treatment of either *entire* structures or *exponentially singular structures* using saddle-point techniques. Apart from periodicity considerations[14] this classification exhausts the class of all labelled iterative structures.

The first basic structure theorem is naturally relative to algebraic-logarithmic structures. We recall that the class $\mathscr{E}$ is defined as the closure of $1, z$ by

$$E(y)=\exp(y), \qquad L(y)=\log(1-y)^{-1}, \qquad Q(y)=(1-y)^{-1}.$$

We have: *Asymptotically, the elementary counting properties relative to strongly aperiodic labelled iterative structures of the algebraic-logarithmic type are expressible rationally in terms of a field of constants $\mathscr{F}_{\mathrm{AL}}$ defined below and of the elements*

$$n!,\ n^{\alpha},\ \log n,\ \rho^{-n},\ \Gamma(-\alpha), \tag{38}$$

with $\alpha, \rho \in \mathscr{F}_{\mathrm{AL}}$. The field $\mathscr{F}_{\mathrm{AL}}$ is the smallest field of constants containing the rationals and the numbers

$$\mathrm{RootOf}[f(\rho)=1], \quad f\in\mathscr{E},$$

and closed under application of functions $g \in \mathscr{E}$.

The indications that we gave in Section 5.3 regarding the classes of entire or exponentially singular functions could be cast in a similar mould. Statements become naturally more cumbersome, and we shall only enunciate a very vague version whose value is only to put saddle-point methods in the proper perspective.

The class of elementary counting properties relative to exponentially singular and entire functions involves algebraic-logarithmic elements plus the class of functions defined as roots of saddle-point equations,

$$\zeta_f(n)\equiv \mathrm{RootOf}\left[\zeta\frac{f'(\zeta)}{f(\zeta)}=n\right], \quad \text{for } f\in\mathscr{E}.$$

[14] Periodicity issues complicate the picture without significantly changing it, so that we do not discuss them in depth in the present paper which is only a short introduction to the subject. Technically, one way of avoiding periodicity problems consists in restricting attention to strongly aperiodic structures in which all component GF's are aperiodic. Notice that periodicity issues are well understood in the context of rational structures, [11, 27, 71].

The class of labelled recursive structures lead to another statement with the constants involving *systems* of equations rather than single equations.

The unlabelled classes naturally lead to yet more complicated formulations, since Pólya operators are involved. We have however explained in Section 5.3 that, in general, their study is of the same mathematical and computational level of difficulty as that of the corresponding labelled classes. (The asymptotic forms remain of the same type, only the field of constants is larger.)

We have summarized in Table 1 some of the classes of problems examined by various authors and discussed throughout this paper. Our approach can thus be viewed as a large programme to generalize and unify a number of results themselves dealing with properties of *classes* of elementary structures.

Zero-one laws and distributions

A parallel enterprise of a generality comparable to ours is that of Compton [22, 20, 23, 21]. Compton starts from so-called 0-1 laws in logic and finite model theory. For instance, (random) finite graphs have a 0-1 law, since any first order property of graphs is either true with asymptotic probability 1 or false with asymptotic probability 1; examples of this situation are that almost all large graphs have 5-cliques, almost no graph has isolated points, etc. Compton is able to describe whole *classes* of logical theories having 0-1 laws. Furthermore his results also cover statistical regularities that are bound to occur in general classes of combinatorial structures, regarding the mean number of components in random structures or the existence of asymptotic probability (not necessarily 0 or 1), in first or second order logical theories.

Another category of results stems from the original observations of Bender and Canfield [4, 14] that certain general combinatorial schemes like sequence or set formation, whose translation into generating functions is

$$\frac{1}{1-uC(z)} \quad \text{or} \quad \exp(uC(z)), \tag{39}$$

Table 1
A classification of some families of structures.

Labelled Iterative structures	**Unlabelled Iterative structures**
• algebraic-logarithmic structures	• regular languages and finite automata
• exponentially singular structures	• general UI structures
• entire structures	
• general LI structures	
Labelled Recursive structures	**Unlabelled Recursive structures**
• quadratic structures	• context-free languages [30]
• algebraic structures; W-structures	• simple families of Meir and Moon [59]
• simple families of Meir and Moon [59]	• graph trees of Harary et al. [42]
• general LR structures	• general UR structures

lead to Gaussian distributions under quite general analytic conditions on the series C. (These schemes give bivariate generating functions for the *distribution* of the number of components in sequence or set constructions.)

Consider the number of cycles in a random permutation or a derangement, the number of components in a random mapping, the number of irreducible factors in a random polynomial over a finite field. The occurrence of a common structural-analytic scheme "explains" the origin of a limiting normal distribution for these rather diverse combinatorial structures [35].

Such questions can be pushed much further and one might aim at a complete characterization of limit distributions that occur inside elementary structures of the LI, UI, LR, UR classes. The problems are naturally more complicated since we are then dealing with bivariate problems.

For recursive structures, we are confronted with non-linear bivariate functional equations. For instance, little is known (to us, at least) on the distribution of path length in plane trees. The bivariate GF of the exact distribution satisfies the non-linear difference equation,

$$F(z; q) = \frac{z}{1 - F(qz; q)},$$

and, though a limit distribution was proved to exist [57], no analytic form seems to be known for the density. In the same vein, the existence of a limit law for the comparison cost of Quicksort was established only recently [68]. This corresponds to the non-linear differential equation

$$\frac{\partial C(z; q)}{\partial z} = C^2(qz; q).$$

Many things are known about the moments [46], but the exact form of the density remains a mystery.

For iterative structures, we deal with explicit functional forms, and there is good hope of approaching a fairly extensive classification of problems. An important step in this direction has been made by Michèle Soria in her thesis [75]. She is able to detect schemes of considerable generality that are associated with the occurrence of diverse limit distributions like Gaussian, Rayley, geometric, Poisson, and the like.

This suggests that a distributional counterpart of our framework should also exist.

As a final conclusion, we are led to believe in the existence of a fascinating domain of investigation. Its scope is the relation between combinatorial structure and asymptotic form and we propose to call it *statistical combinatorics.*

Acknowledgment

This work was supported in part by the Esprit Basic Research Action No. 3075 (Project ALCOM). The authors are grateful to François Morain, Michèle Soria,

and Robert Sedgewick for their comments on the manuscript. The ideas developed here have greatly benefited from innumerable insightful discussions with Jean-Marc Steyaert and Michèle Soria. Maurice Nivat's constant support (and patience during the final labour!) is gratefully acknowledged.

References

[1] A.V. Aho and J.D. Ullman, *The Theory of Parsing, Translation, and Compiling*; *Volume* 1: *Parsing* (Prentice Hall, Englewood Cliffs, NJ, 1972).

[2] L. Albert, R. Casas, F. Fages and P. Zimmermann, Average case analysis of unification algorithms, Research Report 1213, Institut National de Recherche en Informatique et en Automatique, April 1990; also in: *Proc. STACS '91*, Lecture Notes in Computer Science (Springer, Berlin, 1991).

[3] G.E. Andrews, *The Theory of Partitions, Encyclopedia of Mathematics and its Applications, Vol. 2* (Addison-Wesley, Reading, MA, 1976).

[4] E.A. Bender, Central and local limit theorems applied to asymptotic enumeration, *J. Combinatorial Theory* **15** (1973) 91-111.

[5] E.A. Bender and J.R. Goldman, Enumerative uses of generating functions, *Indiana Univ. Math. J.* (1971) 753-765.

[6] F. Bergeron and G. Cartier, Darwin: Computer algebra and enumerative combinatorics, in: R. Cori and M. Wirsing, eds., *STACS-88*, Lecture Notes in Computer Science **294** (Springer, Berlin, 1988).

[7] F. Bergeron, G. Labelle and P. Leroux, Functional equations for data structures, in: R. Cori and M. Wirsing, eds., *STACS-88*, Lecture Notes in Computer Science **294** (Springer, Berlin, 1988).

[8] J. Berstel, Sur la densité asymptotique de langages formels, in: M. Nivat, ed., *Automata, Languages and Programming* (North-Holland, Amsterdam, 1972) 345-358.

[9] J. Berstel, ed., *Séries Formelles* (LITP, University of Paris, 1978).

[10] J. Berstel and C. Reutenauer, Recognizable formal power series on trees, *Theoret. Comput. Sci.* **18** (1982) 115-148.

[11] J. Berstel and C. Reutenauer, *Les Séries Rationnelles et leurs Languages* (Masson, Paris, 1984).

[12] D. Borwein, S. Rankin and L. Renner, Enumeration of injective partial transformations, *Discrete Math.* **73** (1989) 291-296.

[13] N. Bourbaki, Fonctions d'une variable réelle, in: *Eléments de Mathématiques*, 2nd edn. (Hermann, 1951) Chap. 5, 36-55.

[14] E.R. Canfield, Central and local limit theorems for the coefficients of polynomials of binomial type, *J. Combinatorial Theory, Series A* **23** (1977) 275-290.

[15] B. Char, K. Geddes, G. Gonnet, M. Monagan and S. Watt, *MAPLE: Reference Manual* 5th edn. (University of Waterloo, 1988).

[16] N. Chomsky and M.P. Schützenberger, The algebraic theory of context-free languages, in: P. Braffort and D. Hirschberg, eds., *Computer Programming and Formal Languages* (North-Holland, Amsterdam, 1963) 118-161.

[17] C. Choppy, S. Kaplan and M. Soria, Complexity analysis of term rewriting systems, *Theoret. Comput. Sci.* **67** (1989) 261-282.

[18] J. Cohen, Computer-assisted microanalysis of programs, *Comm. ACM* **25**(10) (1982) 724-733.

[19] J. Cohen and T. Hickey, Uniform random generation of strings in a context-free language, *SIAM J. Comput.* **12**(4) (1983) 645-655.

[20] K.J. Compton, A logical approach to asymptotic combinatorics. I. First order properties, *Adv. Math.* **65** (1987) 65-96.

[21] K.J. Compton, A logical approach to asymptotic combinatorics. II. Second-order properties. *J. Combinatorial Theory, Series A* **50** (1987) 110-131.

[22] K.J. Compton, Some methods for computing component distribution probabilities in relational structures, *Discrete Math.* **66** (1987) 59-77.

[23] K.J. Compton, 0-1 laws in logic and combinatorics, in: I. Rival, ed., *Proc. NATO Advanced Study Institute on Algorithms and Order* (Reidel, Dordrecht, 1988) 353-383.

[24] L. Comtet, *Advanced Combinatorics* (Reidel, Dordrecht, 1974).
[25] N.G. de Bruijn, *Asymptotic Methods in Analysis*, 3rd edn. (North-Holland, 1958; reprinted by Dover, 1981).
[26] M.-P. Delest and G. Viennot, Algebraic languages and polyominoes enumeration, *Theoret. Comput. Sci.* **34** (1984) 169–206.
[27] S. Eilenberg, *Automata, Languages, and Machines, Vol. A* (Academic Press, 1974).
[28] W. Feller, *An Introduction to Probability Theory and its Applications*, 3rd edn., Vol. 1 (Wiley, New York, 1968).
[29] P. Flajolet, *Analyse d'Algorithmes de Manipulation d'Arbres et de Fichiers*, Cahiers du Bureau Universitaire de Recherche Opérationnelle, Vols. 34, 35 (Université Pierre et Marie Curie, Paris, 1981) 209 pp.
[30] P. Flajolet, Analytic models and ambiguity of context-free languages, *Theoret. Comput. Sci.* **49** (1987) 283–309.
[31] P. Flajolet and A.M. Odlyzko, Random mapping statistics, in: J.-J. Quisquater, ed., *Proc. Eurocrypt'89* Lecture Notes in Computer Science **434** (Springer, Berlin, 1990) 329–354.
[32] P. Flajolet and A.M. Odlyzko, Singularity analysis of generating functions, *SIAM J. Discrete Math.* **3**(2) (1990) 216–240.
[33] P. Flajolet, B. Salvy and P. Zimmermann, Lambda-Upsilon-Omega: An assistant algorithms analyzer, in: T. Mora, ed., *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, Lecture Notes in Computer Science **357** (Springer, Berlin, 1989) 201–212.
[34] P. Flajolet, B. Salvy and P. Zimmermann, Lambda-Upsilon-Omega: The 1989 Cookbook, Research Report 1073, Institut National de Recherche en Informatique et en Automatique, August 1989, 116 pp.
[35] P. Flajolet and M. Soria, Gaussian limiting distributions for the number of components in combinatorial structures, *J. Combinatorial Theory, Series A* **53** (1990) 165–182.
[36] P. Flajolet and J.-M. Steyaert, A complexity calculus for classes of recursive search programs over tree structures, in: *Proc. 22nd Ann. Symp. on Foundations of Computer Science* (IEEE Computer Society Press, 1981) 386–393.
[37] P. Flajolet and J.-M. Steyaert, A complexity calculus for recursive tree algorithms, *Math. Systems Theory* **19** (1987) 301–331.
[38] D. Foata, *La Série Génératrice Exponentielle dans les Problèmes d'Énumération* (S.M.S. Montreal University Press, 1974).
[39] I.P. Goulden and D.M. Jackson, *Combinatorial Enumeration* (Wiley, New York, 1983).
[40] D.H. Greene, Labelled formal languages and their uses, Ph.D. Thesis, Stanford University, 1983.
[41] F. Harary and E. Palmer, *Graphical Enumeration* (Academic Press, 1973).
[42] F. Harary, R.W. Robinson and A.J. Schwenk, Twenty-step algorithm for determining the asymptotic number of trees of various species, *J. Austral. Math. Soc.* (*Series A*) **20** (1975) 483–503.
[43] G.H. Hardy, Orders of infinity, *Cambridge Tracts Math.* **12** (1910).
[44] B. Harris and L. Schoenfeld, Asymptotic expansions for the coefficients of analytic functions, *Illinois J. Math.* **12** (1968) 264–277.
[45] W.K. Hayman, A generalization of Stirling's formula, *J. Reine Angew. Math.* **196** (1956) 67–95.
[46] P. Hennequin, Combinatorial analysis of quicksort algorithm, *Theoret. Inform. Applic.* **23**(3) (1989) 317–333.
[47] T. Hickey and J. Cohen, Automating program analysis, *J. ACM* **35** (1988) 185–220.
[48] A.T. Jonassen and D.E. Knuth, A trivial algorithm whose analysis is not, *J. Comput. System Sci.* **16** (1978) 301–322.
[49] A. Joyal, Une théorie combinatoire des séries formelles, *Adv. Math.* **42**(1) (1981) 1–82.
[50] R. Jungen, Sur les séries de Taylor n'ayant que des singularités algébrico-logarithmiques sur leur cercle de convergence, *Comment. Math. Helv.* **3** (1931) 266–306.
[51] S. Karlin and H. Taylor, *A First Course in Stochastic Processes*, 2nd edn. (Academic Press, 1975).
[52] D.E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms* (Addison-Wesley, Reading, MA, 1968).
[53] D.E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching* (Addison-Wesley, Reading, MA, 1973).
[54] D.E. Knuth, *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms*, 2nd edn. (Addison-Wesley, Reading, MA, 1981).

[55] D. Le Metayer, Ace: An automatic complexity evaluator, *ACM Trans. Programming Languages Systems* **10**(2) (1988) 248-266.
[56] M. Lothaire, *Combinatorics on Words. Encyclopedia of Mathematics and its Applications, Vol. 17* (Addison-Wesley, Reading, MA, 1983).
[57] G. Louchard, The Brownian excursion: a numerical analysis, *Comput. Math. Applic.* **10**(6) (1984) 413-417.
[58] P. Massaza, G. Mauri, P. Righi and M. Torelli, A symbolic manipulation system for combinatorial structures, in: T. Beth and M. Clausen, eds., *Applicable Algebra, Error-Correcting Codes, Combinatorics and Computer Algebra*, Lecture Notes in Computer Science **307** (Springer, Berlin, 1987).
[59] A. Meir and J.W. Moon, On the altitude of nodes in random trees, *Can. J. Math.* **30** (1978) 997-1015.
[60] F. Morain and J. Olivos, Speeding up the computations on an elliptic curve using addition-subtraction chains, *RAIRO Theoret. Inform. Applic.* **24**(6), to appear.
[61] J. Moses, Algebraic simplification: a guide for the perplexed, *Comm. ACM* **14**(8) (1971) 527-537.
[62] A.M. Odlyzko and L.B. Richmond, Asymptotic expansions for the coefficients of analytic generating functions, *Aequationes Math.* **28** (1985) 50-63.
[63] F.W.J. Olver, *Asymptotics and Special Functions* (Academic Press, New York, 1974).
[64] G. Pólya, Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen, *Acta Math.* **68** (1937) 145-254.
[65] G. Pólya and R.C. Read, *Combinatorial Enumeration of Groups, Graphs and Chemical Compounds* (Springer-Verlag, New York, 1987).
[66] L.H. Ramshaw, Formalizing the analysis of algorithms, Ph.D. Thesis, Stanford University, 1979; also available as Tech. Rep. SL-79-5, Xerox Palo Alto Research Center, Palo Alto, CA.
[67] R.C. Read, A note on the number of functional digraphs, *Math. Ann.* **143** (1961) 109-110.
[68] M. Régnier, A limiting distribution for quicksort, *Theoret. Inform. Applic.* **23**(3) (1989) 335-343.
[69] G.-C. Rota, On the foundations of combinatorial theory, I. Theory of Möbius inversion, *Z. Wahrscheinlichkeitstheorie* **2** (1964) 340-368.
[70] G.-G. Rota, *Finite Operator Calculus* (Academic Press, New York, 1975).
[71] A. Salomaa and M. Soittola, *Automata-Theoretic Aspects of Formal Power Series* (Springer, Berlin, 1978).
[72] B. Salvy, Fonctions génératrices et asymptotique automatique, Research Report 967, Institut National de Recherche en Informatique et en Automatique, 1989, 118 pp.
[73] B. Salvy, Asymptotique automatique et fonctions génératrices, Ph.D. Thesis, Ecole Polytechnique, 1991, in preparation.
[74] N.J.A. Sloane, *A Handbook of Integer Sequences* (Academic Press, New York, 1973).
[75] M. Soria, Méthodes d'analyse pour les constructions combinatoires et les algorithmes, Doctorat d'Etat, Université de Paris-Sud, Orsay, 1990.
[76] R.P. Stanley, Generating functions, in: G.-C. Rota, ed., *Studies in Combinatorics*, M.A.A. Studies in Mathematics **17** (The Mathematical Association of America, 1978) 100-141.
[77] R.P. Stanley, Differentiably finite power series, *Eur. J. Combinatorics* **1** (1980) 175-188.
[78] R.P. Stanley, *Enumerative Combinatorics*, Vol. I (Wadsworth & Brooks/Cole, 1986).
[79] J.-M. Steyaert, Structure et complexité des algorithmes, Thèse d'Etat à l'Université de Paris VII, 1984.
[80] E.C. Titchmarsh, *The Theory of Functions*, 2nd edn. (Oxford University Press, 1939).
[81] J. Vitter and P. Flajolet, Analysis of algorithms and data structures, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Vol. A: Algorithms and Complexity* (North-Holland, Amsterdam, 1990) 431-524.
[82] J. Vuillemin, A unifying look at data structures, *Comm. ACM* **23**(4) (1980) 229-239.
[83] B. Wegbreit, Mechanical program analysis, *Comm. ACM* **18**(9) (1975) 528-539.
[84] P. Weis, M. Aponte, A. Laville, M. Mauny and A. Suárez, *The CAML Reference Manual* (INRIA-ENS, 1987).
[85] J. Wimp and D. Zeilberger, Resurrecting the asymptotics of linear recurrences, *J. Math. Anal. Applic.* **111** (1985) 162-176.
[86] R. Wong, *Asymptotic Approximations of Integrals* (Academic Press, 1989).
[87] M. Wyman, The asymptotic behavior of the Laurent coefficients, *Can. J. Math.* **11** (1959) 534-555.
[88] D. Zeilberger, A holonomic approach to special function identities, 1988, Preprint.
[89] P. Zimmermann, Alas: un système d'analyse algébrique, Rapport de recherche 968, Institut National de Recherche en Informatique et en Automatique, 1989.

[90] P. Zimmermann, Séries génératrices et analyse automatique d'algorithmes, Ph.D. Thesis, Ecole Polytechnique, Palaiseau, France, 1991.

[91] P. Zimmermann and W. Zimmermann, Analysis of programs with semantic conditionals, 1990, in preparation.

[92] W. Zimmermann, Automatische Komplexitätsanalyse von funktionalen Programmen, Ph.D. Thesis, University of Karlsruhe, 1990.