ORDERED CONSTRUCTION OF COMBINATORIAL OBJECTS

BY

MITCHELL ALAN HARRIS

A.B., Cornell University, 1986

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1998

Urbana, Illinois

Abstract

A large number of combinatorial structures can be specified using context free grammars. These grammars can be interpreted as systems of equations on the enumerating generating functions, showing that counting the structures is closely connected to constructing them, yielding a general method for unranking. We describe the connection of enumeration and construction through formal power series, extend the basic operations with permutation groups and analyse them using Pólya ennumeration, give a bijection for graphs, and finally show the relation to solutions of systems of elements from a semiring.

Acknowledgments

In addition to my advisor Nachum Dershowitz, I would like to thank Paul Zimmermann, Eithne Murray, and John Jozwiak for direction and discussion.

Table of Contents

Chapter

List of Tables

List of Figures

List of Symbols

Chapter 1

Introduction

An essential aspect of combinatorics is enumeration. Once a discrete structure has been identified and described one of the most basic questions one can ask is how many different instances of such structures exist. This has led to great accomplishments in finding solutions that run the scale from exact closed form, to recurrences, to asymptotics. What is often left out of such investigations, mostly considered as a trivial or unimportant secondary issue, is the methodological construction of those instances that are being counted. As we will show, this very general problem is not so trivial, is interesting in its own right, and even practical.

Flajolet and Zimmermann [6, 18] have defined a calculus for the description of general classes of combinatorial structures. They use this calculus both to enumerate the structures, that is to calculate the number of objects of a given size, and to uniformly generate random examples of a given size. Their method is to take a description of the class of structures, which is in effect a grammar for recognition of the class as a language, and calculate a corresponding generating function (formal power series) that enumerates the objects by number of terminal symbols. Here, these methods are extended to produce a bijection of such classes of structures with

the natural numbers, allowing the development of ranking and unranking algorithms. First, I will discuss the use of generating functions for enumerating combinatorial structures defined using the constructors union, product, sequence, cycle, and set, in both labeled and unlabeled universes. Second, I will describe algorithms for each of these constructors for unranking, for creating a unique structure for a given integer. Third, I will derive these generating functions and algorithms from a suitable interpretation of Pólya's cycle index (Zyklenzeiger) and its related interpretation in semirings. And finally, I will show the relationship between grammars producing these structures and semiring operations.

Chapter 2

Combinatorial Structures

2.1 Description

The most common combinatorial objects are those associated with configurations of sets and integers. A *combination* of size k from a set of n elements is a subset of those elements; a *permutation* on the same set is list of distinct elements where position in the list matters; a *set partition* is a partition of a set of elements into subsets; an *integer partition* is a set of positive integers that sum to n. These classes can be differentiated by certain properties: ordered, unordered, labeled, unlabeled, invariant under certain operations. Thus a permutation of a set of n objects is an ordered, labeled set; an integer partition of n is an unordered, unlabeled set of sets whose total size is n. A necklace is an ordered set of unlabeled objects that is the same if it is rotated or reflected.

Some combinatorial objects have a recursively defined substructure. There are many varieties of trees: k–ary trees have nodes of degree k; plane trees have ordered sub–trees; hierarchies are rooted trees that have no nodes with exactly one child; context–free grammar derivations

are parse trees for words in context–free grammars. These are all characterized by objects having sub–objects of the same class.

2.2 Specification

We can create many combinatorial structures using construction grammars. At first we have only the primitive elements ϵ for the null element of size 0 and $x, y, z \ldots$ the terminal symbols (or atoms) of size 1. Then we can create larger objects with the following operations: alternate choice (called disjoint union or '+', or Union), concatenation (called product or '$\times$', or Prod), set, sequence, and cycle. For example, the set of binary trees can be constructed with the following recursive operation:

$$B = \mathrm{Union}(1, \mathrm{Prod}(z, \mathrm{Prod}(B, B))) = 1 + z \times B \times B$$

or as a context–free grammar $G = \{T, V, S, P\}$ of terminals, variables, start symbol and productions from $V \to (V + T)^*$

$$T = \{1, z\}, V = \{B\}, S = B,$$
$$P = \left\{ \begin{array}{l} B \to zBB \\ \\ B \to 1. \end{array} \right\}$$

A binary tree can be constructed by a leaf (the 1) or as an interior node (z) and by two recursive subtrees. See Figure 2.1 for all such trees of weight 4 and Figure 2.2 for those of weight 5.

Considered as a grammar, this sort of formal object description can be used to produce random structures. At each union, make a choice. Then recursively create new substructures, stopping when there are no more subobjects to be expanded. Here, the '+' and '$\times$' are simply

$(((Z,Z),Z),Z)$ $((Z,(Z,Z)),Z)$ $((Z,Z),(Z,Z))$ $(Z,((Z,Z),Z))$ $(Z,(Z,(Z,Z)))$

Figure 2.1: The ordered binary trees with 4 leaves

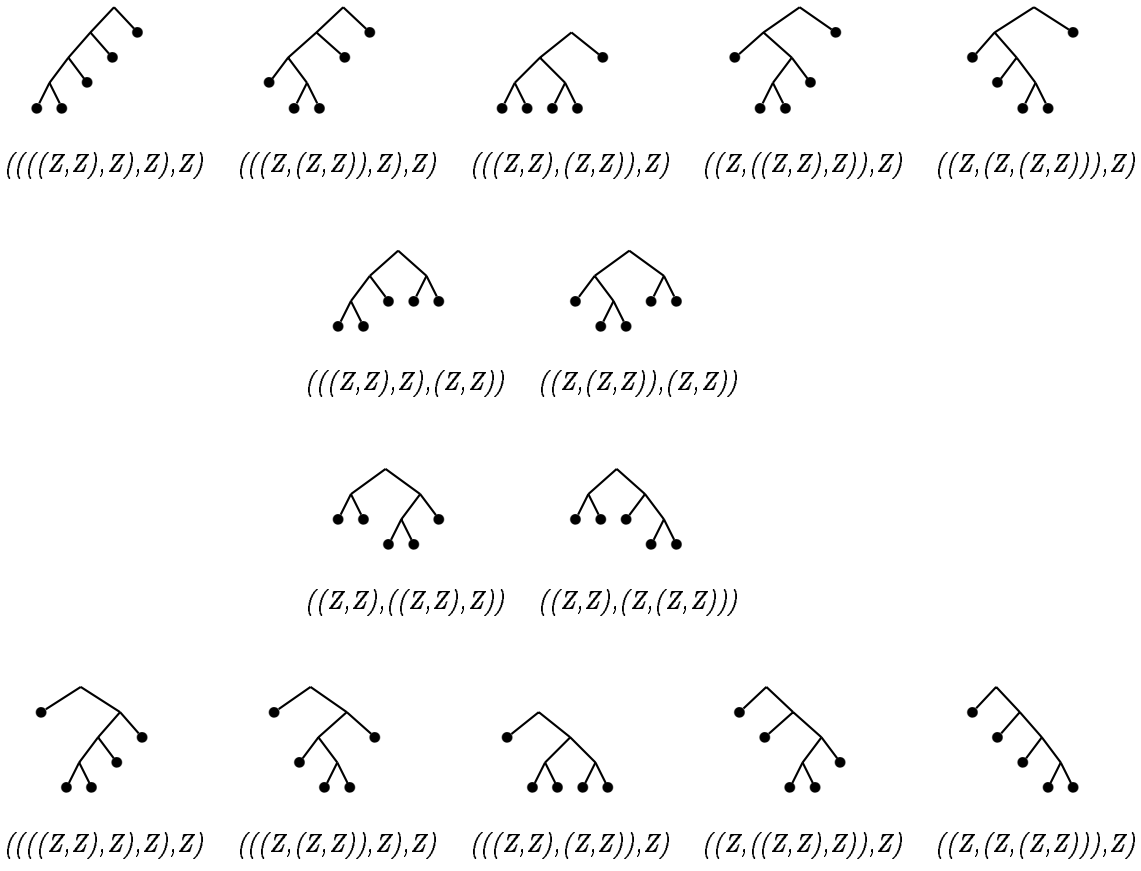

Figure 2.2: The ordered binary trees with 5 leaves

$0, 1$	the identity objects for union and concatenation
$x, y, z \ldots$	objects of size one
Union $(+)$	the union of two disjoint classes
Prod $(\times)$	the Cartesian product of 2 classes
Seq	all the ordered collections of any length
Cyc	same as Seq but only those isomorphic by rotating the elements
Dih	same as Cyc but only those isomorphic by reversing the elements (a dihedral, or cycle that can be 'flipped over')
Set	the power set of items from a class (all the unordered collections of any size with no repetitions)
Bag	the multisets of items from a class (all the unordered collections of any size with repetitions allowed)

Table 2.1: Operations on combinatorial structures

appropriate symbolic shorthand for alternative and concatenation. The object z is a constant symbol (or equivalently an atom) standing for an object of size or weight one. The object **1** is a constant of weight 0 whose concatenation yields the original. The basic objects and operations on structures are in Table 2.1, extended from Zimmermann's work [18, 6].

There are two interpretations of atoms, labeled and unlabeled, which lead to two uses of the constructors. With the labeled interpretation, all atoms in an object are marked uniquely from a set of so that they are all considered distinct and therefore no structure parts can be repeated exactly. This means that in the labeled universe there are no multisets. This is somewhat

analogous to sampling without replacement. Unlabeled atoms yield structures where atoms can only be distinguished by name of the atom.

Some simple extensions are possible. Limitations of size can be made for the iterative constructors; e.g. sequences of 'z' of length less than 5 are constructed by $\mathrm{Seq}(z, \mathrm{card} < 5)$. As above for binary trees, recursion can be used.

Examples of some basic combinatorial labeled structures and their construction grammars are in Table 2.2 and unlabeled in Table 2.3.

permutations	$\mathrm{Seq}(z) = \mathrm{Set}(\mathrm{Cyc}(z))$	the mapping or cycle decomposition
set partitions	$\mathrm{Set}(\mathrm{Set}(z, \mathrm{card} \geq 1))$	set of nonempty sets
surjections	$\mathrm{Seq}(\mathrm{Set}(z, \mathrm{card} \geq 1))$	the onto function is from the nth set in the sequence to n

Table 2.2: Some labeled structures

A labeled example is the set partitions. There are 15 of weight 4 in the customary lattice Π_4 shown in Figure 2.3. The labels for z are all that is shown (e.g. the first one is really $\{\{z_0, z_1, z_2, z_3, z_4\}\}$), since there is only one atom.

An unlabeled example is the 2–combinations from a set of 4 are constructed from $(1+z)^4$; they are 4–tuples whose elements can be either z or 1 whose total weight is 2. They are:

$$<z,z,1,1> \quad <z,1,z,1> \quad <z,1,1,z> \quad <1,z,z,1> \quad <1,z,1,z> \quad <1,1,z,z>$$

Combinations are usually viewed as 'labeled' objects:

$$\{1,2\} \quad \{1,3\} \quad \{1,4\} \quad \{2,3\} \quad \{2,4\} \quad \{3,4\}.$$

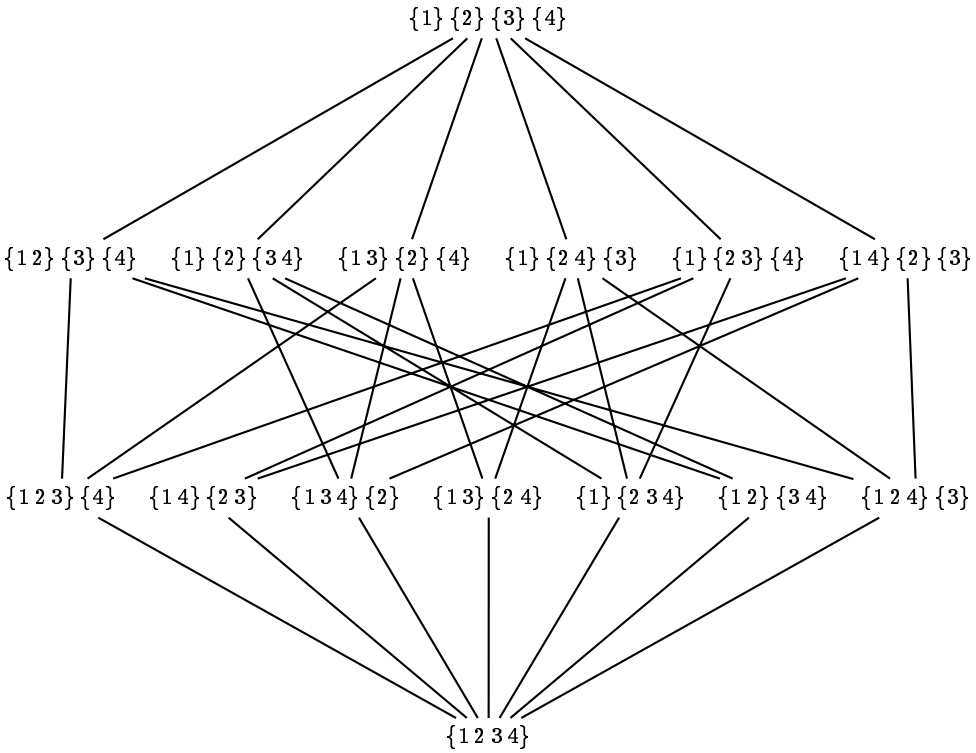

Figure 2.3: Set partitions of size 4

r–combinations	$(1+z)^n$	from a set of n elements, an element is either in a subset (z), or not (1)
integer partitions	$\mathrm{Bag}(\mathrm{Bag}(z, \mathrm{card} \geq 1))$	
rooted trees (nested multisets)	$\mathrm{RT} = z \times \mathrm{Bag}(\mathrm{RT})$	z is the root with an unordered set of subtrees
rooted plane trees	$\mathrm{RPT} = z \times \mathrm{Seq}(\mathrm{RPT})$	z is the root with an ordered set of subtrees restricted to the plane, i.e. a sequence
irreversible necklaces	$\mathrm{Cyc}(\mathrm{Bag}(z, \mathrm{card} \geq 1))$	cycles
necklaces	$\mathrm{Dih}(\mathrm{Bag}(z, \mathrm{card} \geq 1))$	dihedrals

Table 2.3: Some unlabeled structures

What we mean by labeled is that every atom is labeled distinctly. In this latter traditional example, the atoms are 1,2,3, and 4, with the unavoidable confluence of notation between integer labels and atoms with integer name. 2–combinations from a set of size 4 can also be constructed to look the traditional way with $A = \mathrm{Set}(x + y + z + w, \mathrm{card} = 2)$.

In the labeled universe, Seq, Cyc, Dih, and Set can all be constructed from $+$ and $\times$ as in Table 2.4.

$$\begin{aligned}
\mathrm{Seq}(A) &= 1 + A + A \times A + A \times A \times A + \cdots \\
&= \sum_{0 \le i} A^i \\
\mathrm{Cyc}(A) &= A + \frac{1}{2} A \times A + \frac{1}{3} A \times A \times A + \frac{1}{4} A \times A \times A \times A + \cdots \\
&= \sum_{1 \le i} \frac{1}{i} A^i \\
\mathrm{Dih}(A) &= A + \frac{1}{2} A \times A + \frac{1}{3 \cdot 2} A \times A \times A + \frac{1}{4 \cdot 2} A \times A \times A \times A + \cdots \\
&= \frac{1}{2} \sum_{1 \le i} \frac{1}{i} A^i + \frac{1}{2} A + \frac{1}{4} A \times A \\
&= \frac{1}{2} \mathrm{Cyc}(A) + \frac{1}{2} A + \frac{1}{4} A \times A \\
\mathrm{Set}(A) &= 1 + A + \frac{1}{2!} A \times A + \frac{1}{3!} A \times A \times A + \frac{1}{4!} A \times A \times A \times A + \cdots \\
&= \sum_{0 \le i} \frac{1}{i!} A^i
\end{aligned}$$

Table 2.4: Generating functions for labeled structures

For example, the four operations are shown for labeled objects of size four in Table 2.5.

To enumerate these classes, we use the exponential generating function whose nth coefficient is the number of objects with n atoms.

size	permutations	cycles	necklaces	combinations
1	(1)	(1)	(1)	(1)
2	(1 2) (2 1)	(1 2)	(1 2)	(1 2)
3	(1 2 3) (2 1 3) (3 1 2) (1 3 2) (2 3 1) (3 2 1)	(1 2 3) (1 3 2)	(1 2 3)	(1 2 3)
4	(1 2 3 4) (2 1 3 4) (3 1 2 4) (4 1 2 3) (1 2 4 3) (2 1 4 3) (3 1 4 2) (4 1 3 2) (1 3 2 4) (2 3 1 4) (3 2 1 4) (4 2 1 3) (1 3 4 2) (2 3 4 1) (3 2 4 1) (4 2 3 1) (1 4 2 3) (2 4 1 3) (3 4 1 2) (4 3 1 2) (1 4 3 2) (2 4 3 1) (3 4 2 1) (4 3 2 1)	(1 2 3 4) (1 2 4 3) (1 3 2 4) (1 3 4 2) (1 4 2 3) (1 4 3 2)	(1 2 3 4) (1 2 4 3) (1 3 2 4)	(1 2 3 4)
total	24	6	3	1

Table 2.5: Labeled structures similar to permutations

There is a subtle notational "confusion" here between generating function and constructor, most noticeably in the specification for combinations. As will be shown later, this overuse of notation is intentional.

These rules for constructing these objects can be used for determining the number of possible objects with a given number of atoms (or terminal symbols). A generating function for a class of objects is simply the power series on a variable x whose coefficient a_i for x^i is the number of objects with i atoms:

$$A(x) = a_0 + a_1 x + a_2 x^2 + \cdots = \sum_{i=0}^{\infty} a_i \cdot x^i$$

where a_i is the number of A objects of size i. Algebraic operations on power series are well defined; they correspond with the combinatorial operations given above. Given two combinatorial classes, A and B, whose enumerating generating functions are $A(z)$ and $B(z)$ respectively, if the class C can be constructed from A and B then the generating function $C(z)$ can be computed as follows:

operation		generating function
$C = A + B$	$\Longrightarrow$	$C(z) = A(z) + B(z)$
$C = A \times B$	$\Longrightarrow$	$C(z) = A(z) \cdot B(z)$
$C = \mathrm{Seq}(A)$	$\Longrightarrow$	$C(z) = \dfrac{1}{1 - A(z)}$
$C = \mathrm{Cyc}(A)$	$\Longrightarrow$	$C(z) = \log \dfrac{1}{1 - A(z)}$
$C = \mathrm{Dih}(A)$	$\Longrightarrow$	$C(z) = \dfrac{1}{2} \log \dfrac{1}{1 - A(z)} + \dfrac{1}{2} A(z) + \dfrac{1}{4} A(z)^2$
$C = \mathrm{Set}(A)$	$\Longrightarrow$	$C(z) = \exp(A(z))$

The derivations of '$+$' and '$\times$' come from simple combinatorics, and with little surprise, the use of the symbols '$+$' and '$\times$' for both operations on strings and operations on polynomials is not coincidental, those for Seq, Cyc, Dih, and Set from the Taylor series expansion of their

respective functions. The definitions for Seq, Cyc, Dih, and Set are mostly useful as shorthand for the infinite series and do not figure directly in actual computations, though the Taylor series expansions do.

Note the bounds on the summation for the expansions. These make combinatorial sense. For Seq and Set, a sequence or set of length zero, an object of weight zero exists, namely **1**, because the only property it has is ordered or unordered. For Cyc and Dih, an object must be cycled or reversed to check for isomorphism, but the empty object cannot be cycled or reversed.

Given the formal description of ordered binary trees, we can either compute the coefficients directly or calculate symbolically a closed form of the generating function from the recurrence relation using conventional techniques:

$$\begin{aligned}
B(z) &= 1 + zB(z)^2 = 1 + z + 2z^2 + 5z^3 + 14z^4 + \cdots \\
0 &= zB(z)^2 - B(z) + 1 \\
B(z) &= \frac{1 - \sqrt{1-4z}}{2z} \\
[z^n]B(z) &= [z^{n-1}]B(z) \cdot (-4) \cdot \frac{(2n-1)}{2} \cdot \frac{1}{n} \\
&= \frac{1}{(n+1)!} \frac{2^n \cdot (2n)!}{2^n \cdot n!} \\
&= \frac{1}{(n+1)} \binom{2n}{n} = C(n)
\end{aligned}$$

where the coefficients of $B(z)$ are the Catalan numbers C(n).[1] (Note: $[z^n]B(z)$ is the coefficient of z^n in the power series $B(z)$.) In general, the generating function can be finitely approximated

[1] Combinatorially, the following grammar constructs the same structures though with leaf nodes of weight 1:

$$B' = \text{Union}(z, \text{Prod}(B', B')) = z + B' \times B'$$

$$T = \{z\}, V = \{B\}, S = B, P = \left\{ \begin{array}{l} B \to B'B' \\ \\ B' \to z \end{array} \right\}$$

By analysis of its characteristic equation, $B(z)^2 - B(z) + z = 0$, we come up with a closed form $[z^{n+1}]B'(z) = C(n)$.

from such recursively defined descriptions by repeated application of the operations to an initial value, stopping when the number of iterations have reached the maximum number of objects wanted.

For unlabeled structures, where atoms are not marked uniquely, enumerating generating functions can be computed from the ordinary generating function. Given A and B, whose ordinary generating functions are $A(z)$ and $B(z)$ respectively, if the class C can be constructed from A and B then the generating function $C(z)$ can be computed as in Table 2.6.

$$
\begin{array}{lcll}
C = A + B & \Longrightarrow & C(z) & = A(z) + B(z) \\
C = A \times B & \Longrightarrow & C(z) & = A(z) \cdot B(z) \\
C = \mathrm{Seq}(A) & \Longrightarrow & C(z) & = \dfrac{1}{1 - A(z)} \\
\\
C = \mathrm{Cyc}(A) & \Longrightarrow & C(z) & = \displaystyle\sum_{1 \le i} \frac{\phi(i)}{i} \log \frac{1}{1 - A(z^i)} \\
\\
C = \mathrm{Dih}(A) & \Longrightarrow & C(z) & = \dfrac{1}{2}\mathrm{Cyc}(A(z)) + \dfrac{1}{4}(2A(z) + A(z^2) + A(z)^2)\,\mathrm{Seq}(A(z^2)) \\
C = \mathrm{Bag}(A) & \Longrightarrow & C(z) & = \exp\left(\dfrac{A(z)}{1} + \dfrac{A(z^2)}{2} + \dfrac{A(z^3)}{3} + \cdots\right) \\
 & & & = \exp(\displaystyle\sum_{1 \le i} \frac{1}{i} A(z^i)) \\
\\
C = \mathrm{Set}(A) & \Longrightarrow & C(z) & = \exp(A(z) - \dfrac{A(z^2)}{2} + \dfrac{A(z^3)}{3} - \cdots) \\
 & & & = \exp(\displaystyle\sum_{1 \le i} (-1)^{i+1} \frac{1}{i} A(z^i))
\end{array}
$$

Table 2.6: Generating functions for unlabeled structures

$\phi(n)$ is Euler's totient function, the number of positive integers less than and relatively prime to n. Only the first three correspondences are obvious; the combinatorial explanation of the rest will be shown in the next chapter. These equations can be found in Zimmermann's papers [6, 18]; a new derivation of them is given here.

A good example of the connections among the constructor, the generating function, and the corresponding class of combinatorial objects is the Fibonacci sequence. Take the basic combinatorial proof: the number of distinct ways F_n of placing 1's and 2's in a row whose sum is n is equal to such rows of length $n-1$ followed by a 1 or rows of length $n-2$ followed by a 2, and these two cases are mutually exclusive. This yields the recurrence relation:

$$F_n = F_{n-1} + F_{n-2}, F_0 = 1, F_1 = 1$$

(we define $F_0 = 1$ because by combinatorial convention there is exactly one way to do nothing). To solve this linear recurrence numerically in the usual way, we find the roots of the characteristic equation and get $F_n = 1/\sqrt{5}(\phi^n - \phi^{-n})$, where $\phi = (1-\sqrt{5})/2$. ϕ To get the generating function $F(x)$ from the recurrence, we need to do the following:

$$\begin{aligned} F(x) &= F_0 + F_1 x + F_2 x^2 + \cdots \\ xF(x) &= F_0 x + F_1 x^2 + \cdots \\ x^2 F(x) &= F_0 x^2 + \cdots \end{aligned}$$

then solving this system for $F(x)$:

$$\begin{aligned} F(x) - F(0) - xF(1) &= -xF(0) + xF(x) + x^2 F(x) \\ F(x) - xF(x) - x^2 F(x) &= F(0) + x(F(1) - F(0)) \\ F(x) &= \frac{1}{1 - x - x^2} \end{aligned}$$

Instead, we can use the combinatorial constructors as follows. We have singletons or pairs of unlabeled elements (size 1 or 2) that are placed in a sequence. This yields immediately:

$$\mathrm{Seq}(x + x^2) = \frac{1}{1 - (x + x^2)}.$$

So we see the generating function is also the object generator. The closed form for the recurrence is convenient for calculating numbers; the generating function is convenient for calculating constructions.

These methods show that from a combinatorial description of a class, we can quickly get a symbolic form for the generating function. Using the algebra of formal power series, for each of the combinatorial operators we can compute the actual coefficients in the power series. For example, we can compute the generating function of $1/(1 - A(x))$, where $A(x)$ is a polynomial.[2]

A short example, one already in Table 2.3, is that for set partitions. Combinatorially, the class is generated by taking labeled sets of sets of size greater than one. Translating directly to the generating function yields $e^{e^x - 1}$, the classic generating function for the Bell numbers, which is what we expect.

To be explicit, a construction using these methods only produces implementations isomorphic to the intended structure, but not necessarily a unique implementation. For example, the two constructors:

$$\begin{aligned} \mathrm{Perm1} &= \mathrm{Seq}(z) \\ \mathrm{Perm2} &= \mathrm{Set}(\mathrm{Cyc}(z)) \end{aligned}$$

[2]If $A(x)$ has a non–zero coefficient a_0, then, in $\mathrm{Seq}(A(z))$, then the repeated multiplication of $A(x)$ will always make additions to every coefficient of B_0. Since the series is unbounded, each coefficient is unbounded also. So we can only make the iterative constructors on objects whose generating function has $f_0 = 0$. This restriction also applies to the other infinite operations.

are combinatorially isomorphic; they both represent the set of all permutations. The first is the implementation as a functional map, the second as a cyclic decomposition. It is easy to see that the generating functions are identical:

$$\begin{aligned} \mathrm{Perm1}(z) &= 1/(1-z) \\ \mathrm{Perm2}(z) &= \exp(\log(1/(1-z))) = 1/(1-z). \end{aligned}$$

Another interesting simplification is that for unlabeled $\mathrm{Set}(z)$, the sets on an atom without repetition. Combinatorially, it is obvious that $\mathrm{Set}(z) = 1+z$: the only sets constructible out of a singleton is the empty set or the singleton itself. The generating function also bears this out:

$$\begin{aligned} \mathrm{Set}(z) &= \exp(\sum_{i\geq 1}(-1)^i \frac{1}{i} z^i) \\ &= \exp(\log(1+z)) \qquad \text{from the Taylor expansion} \\ &= 1+z. \end{aligned}$$

Chapter 3

Permutation Groups and Structure Isomorphism

3.1 Permutations

Objects, as we have described, that have structured components can be considered as permutations on items from a set, where some permutations are considered the same as others. Suppose these equivalences are determined by a set of permutations on the same number of objects. The notation

$$\backslash x_1\, x_2\, \cdots x_n \backslash$$

is a map;

$$(x_1\, \ldots) \ldots (\ldots\, x_n)$$

is a cycle decomposition (usually with singleton cycles left out). Given the permutation $\backslash 2\, 3\, \ldots\, n\, 1 \backslash$, a full cycle on all the elements, the permutations $(1\, 2\, \cdots\, n)$ is equivalent to $(2\, 3\, \cdots\, n\, 1)$. By transitivity, it is also equivalent to $(3\, 4\, \cdots\, n\, 1\, 2) \cdots (n\, 1 \cdots n-1)$. This is

the same description of cycles in terms of isomorphism classes induced by a group of permutations. A permutation group G acting on a set S (which we normally take to be the set $[n] = \{1, 2, \ldots, n\}$) is a set of permutations on S closed on composition.

Since any set of permutations on n objects generates a subgroup of the symmetric group of order n, we will start with the properties of this group in order to derive those of its subgroups. A permutation group G acting on the set S (we only consider $S = [n] = \{0, 1, 2, \ldots n-1\}$) defines equivalence classes on the set S. If a permutation takes one object to another then we say they are isomorphic. Objects that are distinct after action by G are the non-isomorphic objects we seek; a representative from the quotient set (or cosets) S/G is chosen from each. By Lagrange's Theorem, $|S/G| = |S|/|G|$. To illustrate, the dihedral group on n objects is of order $2n$. The number of distinct permutations induced then is $\frac{n!}{2n} = \frac{(n-1)!}{2}$. Knowing the order of a group or family of groups, we can easily compute its enumerating generating function. Constructing objects from classes generated from arbitrary permutation groups is a little more involved. A permutation group G acting on a set of size n induces a quotient set on S_n.[1] The members of each of the cosets of S_n/G are now equivalent over the permutations G; all we seek is a single representative from each coset.

3.2 Pólya Enumeration

Pólya's methods of enumeration is used to compute generating functions for objects with invariance under permutations. Over all permutations of n objects, there is an equivalence relation

[1] A fact from basic group theory is that if G is normal in S_n then the quotient set S_n/G is also a group and that there is a unique homomorphism from S to S_n/G and G maps to the identity element of S_n/G. Since we will deal with arbitrary groups we are not guaranteed this structure on the cosets.

on the set acted on by a permutation. For example, on the set $[n] = \{0, 1, \ldots, n-1\}$, the single permutation cycle $(0\,1\,\cdots\,n-1)$, leaves $(n-1)!$ equivalence classes of permutations, since there are n similar permutations by applying the cycle up to n times. With this in mind, we seek to enumerate these equivalence classes for a given group of permutations to get the non–isomorphic objects under the action of that group. Since a set of permutations generates a group, we can use some group concepts to aid in finding the generating function.

A basic notion is Burnside's Lemma that the number of equivalence classes, $N(G)$, induced by a permutation group G acting on a set S is:

$$N(G) = \frac{1}{|G|} \sum_{\pi \in G} \mathrm{FP}(\pi)$$

where $\mathrm{FP}(\pi)$ is the number of fixed points in a permutation π, that is, the number of cycles of length one. To show this, consider the set S that G acts on. For $s \in S$, s equals $\pi(s)$ for some π, and certainly at least one in the identity permutation. The number of fixed points over all permutations must equal the number of times that $s = \pi(s)$ over all items in S. If we call $\mathrm{FP}(s)$ the number of permutations π where $s = \pi(s)$, then combinatorially, by first counting the fixed points over all permutations and then counting them over all elements of S, we get:

$$\sum_{\pi \in G} \mathrm{FP}(\pi) = \sum_{s \in S} \mathrm{FP}(s)$$

The cycle decompositions of the permutations in a group G can now be transformed directly to a generating function: for a permutation acting on a set enumerated by another generating function $A(x)$, each cycle of length k in a permutation contributes a factor of $A(x^k)$ (this is sometimes abbreviated as a distinct variable x_k). The generating function for the objects enumerated by $A(x)$ acted upon by a permutation group G is then:

$$Z(G)[A(x)] = \frac{1}{|G|} \sum_{\pi \in G} \prod_{\text{cycle } \pi' \in \pi} A(x^{|\pi'|})$$

Since this function depends on the cycle decomposition of the permutations, we can write this using a slightly different notation, where the permutations in G are represented by the integer partition

$$Z(G)[A(x)] = \frac{1}{|G|} \sum_{\pi \in G} \prod_{k=1}^{|G|} A(x^k)^{e_k}$$

where e_k is the number of cycles of length k in the permutation π.

For the more common families of groups, we can compute the cycle index as follows (following Harary and Palmer [8]):

- Identity group I_n. Sometimes called E_n, this is the trivial group whose permutation group is defined by the single permutation $(0)(1)\ldots(n-1)$ whose generating term is x_1^n. No collection of elements is equivalent to any other unless it has exactly the same elements.

- Cyclic group C_n. Generated by the single permutation $\pi = (1\,2\,\ldots\,n-1\,0)$. If n is not prime, then some of the permutations repeat when generated as π^k. Specifically, for any divisor d of n, there will be $\phi(d)$ permutations that have exactly n/d cycles of length d. This yields the term $\phi(d)x_d^{n/d}$.

- Dihedral group D_n. Generated by $(1\,2\,\ldots\,(n-1)\,0)$ and $((n-1)\,(n-2)\,\ldots\,1\,0)$. It has the same terms as for the cyclic group plus those induced by this extra permutation. If n is odd, we get $(n-1)/2$ pairs and a singleton (or $x_1 x_2^{(n-1)/2}$). If n is even, we likewise have two singletons and $(n-2)/2$ pairs, and also $n/2$ pairs because of the even parity (or $x_1^2 x_2^{(n-2)/2} + x_2^{n/2}$).

- Symmetric group S_n,or Sym(n). Generated by $(0\,1)$ and $(0\,1\,\cdots\,n-1)$. This includes every possible permutation. The cycle structure is defined by all the integer permutations

on n since the cycle decomposition partitions the permutation into disjoint sets. For each partition, the number of permutations in S_n is a function of the different sizes of parts in the partition as a reduction of the maximum number of permutations $n!$. For a single part of size k (corresponding to one cycle of length k in the permutation), the number of permutations is divided by k. If there are j cycles of length k we must divide by $j!$ since each cycle can be permuted with another of the same length. This yields $\frac{n!}{k^{e_k} e_k!}$ for each part of size k, where e is a partition of n, e_i is the number of cycles of length i, and $\sum_{1 \le i \le n} e_i = n$.

- Alternating group A_n, or $\mathrm{Alt}(n)$. The alternating group is only the even permutations, those with an even number of transpositions (swaps or 2–cycles). Only odd cycles are equivalent to an even number of transpositions, so a permutation is odd if it has an odd number of even cycles. Using this, we can take the cycle index for S_n and account for even cycles.

This is summarized in Table 3.1 (following [8]) where $\lambda(n)$ is the set of integer partitions of n: $\langle e_1, e_2, \ldots, e_n \rangle \in \lambda(n), n = \sum_{k=1}^{n} k e_k$.

For an arbitrary permutation group, we can compute the cycle index naively by listing all the group elements and adding its appropriate generating function term, using basic group algorithms as in Butler [3] and permutation algorithms in Nijenhuis and Wilf [12]. This naive algorithm is as good as we can get; Goldberg [7] has shown this problem to be NP–complete. Luckily, most applications are limited to those groups mentioned above.

$$\text{Identity:} \quad Z(I_n) = x_1^n \tag{3.1}$$

$$\text{Cyclic:} \quad Z(C_n) = \frac{1}{n}\sum_{d|n}\phi(d)x_d^{n/d} \tag{3.2}$$

$$\text{Dihedral:} \quad Z(D_n) = \frac{1}{2}Z(C_n) + \begin{cases} \frac{1}{2}x_1 x_2^{(n-1)/2} & \text{odd} \\ \frac{1}{4}\left(x_2^{n/2} + x_1^2 x_2^{(n-2)/2}\right) & \text{even} \end{cases} \tag{3.3}$$

$$\text{Symmetric:} \quad Z(S_n) = \frac{1}{n!}\sum_{e\in\lambda(n)}\prod_{1\le k\le n}\frac{(x_k/k)^{e_k}n!}{e_k!}$$

$$= \sum_{e\in\lambda(n)}\prod_k \frac{x_k^{e_k}}{k^{e_k}e_k!} \tag{3.4}$$

$$\text{Alternating:} \quad Z(A_n) = \sum_{e\in\lambda(n)}\prod_k \frac{1+(-1)^{e_2+e_4+\cdots}x_k^{e_k}n!}{k^{e_k}e_k!} \tag{3.5}$$

Table 3.1: Group cycle–index generating functions

As an example, Figure 3.1 shows the objects of size four produced by

$$\begin{aligned} C_4[x+y] &= \tfrac{1}{4}(s_1^4 + s_2^2 + 2s_4)[x+y] \\ &= \tfrac{1}{4}((x+y)^4 + (x^2+y^2)^2 + 2(x^4+y^4)) \\ &= x^4 + x^3y + 2x^2y^2 + xy^3 + y^4. \end{aligned}$$

Combinatorially this is equivalent to the cycles on four points of two colors.

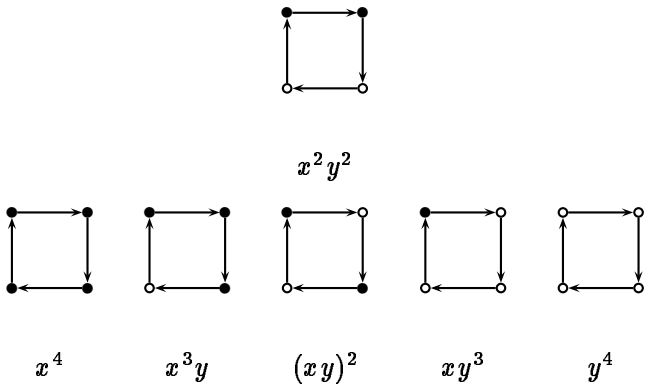

Figure 3.1: The cycles on four points of two colors

Harary and Palmer [8] show that operations on permutation groups yield operations on the cycle index. The direct product of two groups gives the product of the two generating functions. The lexicographic composition (or wreath product) of two groups gives the composition of their generating functions.

From the cycle indexes on these groups we can determine the cycle index for the family of structures of unspecified size under each group. These will then correspond to the operations above that produce objects that are invariant under a group for any size sub–objects. Here is the new derivation of the generating functions for the combinatorial operators Seq, Cyc, Dih, Set, and Bag:

$$\mathrm{Seq} = \sum_{n\geq 0} Z(I_n) = \sum_{n\geq 0} x_i \frac{1}{(1-x)} \tag{3.6}$$

$$\begin{aligned}
\mathrm{Cyc} = \sum_{n\geq 1} Z(C_n) &= \sum_{n\geq 1} \frac{1}{n} \sum_{d|n} \phi(d) x_d^{n/d} \\
&= \sum_{d\geq 1} \sum_{r\geq 1} \frac{1}{dr} \phi(d) x_d^r \\
&= \sum_{d\geq 1} \frac{\phi(d)}{d} \sum_{r\geq 1} \frac{x_d^r}{r} \\
&= \sum_{d\geq 1} \frac{\phi(d)}{d} \log \frac{1}{1-x_d}
\end{aligned} \tag{3.7}$$

$$\begin{aligned}
\mathrm{Dih} = \sum_{n\geq 1} Z(D_n) &= \frac{1}{2} \sum_{n\geq 1} Z(C_n) + \frac{1}{2} \sum_{n=2k-1\geq 1} x_1 x_2^{(n-1)/2} \\
&\quad + \frac{1}{4} \sum_{n=2k\geq 2} (x_2^{n/2} + x_1^2 x_2^{n/2-1}) \\
&= \frac{1}{2} \mathrm{Cyc}(x) + \frac{1}{4} \sum_{k\geq 1} (2x_1 x_2^{k-1} + x_2^k + x_1^2 x_2^{k-1}) \\
&= \frac{1}{2} \mathrm{Cyc}(x) + \frac{1}{4} (2x_1 + x_2 + x_1^2) \sum_{k\geq 1} x_2^{k-1} \\
&= \frac{1}{2} \mathrm{Cyc}(x) + \frac{1}{4} (2x_1 + x_2 + x_1^2) \sum_{k\geq 0} x_2^k \\
&= \frac{1}{2} \mathrm{Cyc}(x) + \frac{1}{4} (2x_1 + x_2 + x_1^2) \frac{1}{1-x_2}
\end{aligned}$$

$$= \frac{1}{2}\operatorname{Cyc}(x) + \frac{1}{4}(2x_1 + x_2 + x_1^2)\operatorname{Seq}(x^2) \tag{3.8}$$

$$\begin{aligned}
\operatorname{Bag} = \sum_{n\geq 0} Z(S_n) &= \sum_{n\geq 0}\sum_{e\in\lambda(n)}\prod_k \frac{x_k^{e_k}}{k^{e_k}e_k!} \\
&= \sum_{e_1\geq 0}\frac{x_1^{e_1}}{1^{e_1}e_1!}\cdot\sum_{e_2\geq 0}\frac{x_2^{e_2}}{2^{e_2}e_2!}\cdots \\
&= \prod_{i\geq 1}\sum_{e_i\geq 0}\frac{x_i^{e_i}}{i^{e_i}e_i!} \\
&= \prod_{i\geq 1}\sum_{t\geq 0}\frac{(x_i/i)^t}{t!} \\
&= \prod_{i\geq 1} e^{\frac{x_i}{i}} = \exp(\sum_{i\geq 1}\frac{x_i}{i}) \qquad (3.9) \\
\operatorname{Set} = \sum_{n\geq 0} Z(A_n - S_n) &= \exp(\sum_{i\geq 1}(-1)^{i+1}\frac{x_i}{i}) \qquad (3.10)
\end{aligned}$$

Again, these have the parameters to the generating function left out for readability. For example, Bag should really read

$$\operatorname{Bag}(A(x)) = \exp(\sum_{i\geq 1}\frac{A(x^i)}{i})$$

Equation 3.10 is not as straightforward as the rest. A multiset can have repetitions, but a set cannot. So a set is a one–to–one function to the underlying objects. We can get this from $A_n - S_n$ as follows. S_n counts the set of all functions on n. A_n is the set of all even permutations. Its generating function counts all one–to–one functions twice (because each functional pair is counted twice for each value by an even permutation) and all others once. So the difference gives the one–to–one functions:

$$\begin{aligned}
Z(A_n - S_n) &= Z(A_n) - Z(S_n) \\
&= \sum_{e\in\lambda(n)}\prod_k \frac{(-1)^{e_2+e_4+\cdots}x_k^{e_k}n!}{k^{e_k}e_k!}
\end{aligned}$$

Equation 3.10 then follows.

The generating function that results from the application of any of these functions to another generating function will result in a new generating function that counts the corresponding combinatorial object. The cycle indexes for the groups count objects invariant over that particular group and size; the corresponding infinite sum counts objects over all sizes. Two comments can be made here. First, if the base generating function[2] has a nonnegative coefficient for x^0 and one of the infinite sums is applied, then the application will not be convergent. Second, in application of this technique to other groups, not every type of group is amenable to summation; the sporadic simple groups (such as the Mathieu groups) have no infinite ranged parameters; their permutation groups have a specific set of generating permutations that are not parameterizable. This does not prevent one from using the cycle index for those groups as a generating function for a constant sized set. Third, all the groups so far treat all the elements of the underlying set symmetrically, in the sense that the sizes of the orbits of every element are equal. This is not a necessary restriction.

3.3 Enumeration and Construction Algorithms

Here we develop new algorithms to enumerate *and* construct objects defined by these operations efficiently.

3.3.1 Polynomial Operations

All the operations on formal power series can be implemented with polynomials, which can be defined as formal power series where there exists an n, called the *degree*, such that all coefficients with index greater than n are zero. All the operations converge, with some restriction on

[2] Also called the "figure counting series"

values. If a coefficient a_n is needed, all the power series involved in the calculation of a_n need polynomials of degree no more than n to guarantee convergence up to that coefficient. So all running time analyses will be in terms of n, the maximum coefficient desired. Note that we are not considering the operations performed on the integer coefficients for two reasons. First, such operations can be handled by an unlimited precision math package for efficiently computing addition, multiplication, etc. Second, there is no method here to predict a priori the asymptotic complexity of a sequence.

The elementary operations of addition, subtraction, creation of zero and identity, are $\Theta(n)$, because they all access or set every coefficient of a polynomial exactly once. Multiplication of polynomials is $\Theta(n^2)$; the exact formula is:

$$h_i = \sum_{k=0}^{i} f_k g_{i-k}.$$

Here a three–for–four algorithm will not aid us because we are not dealing with symbolic polynomials; all operations ignore coefficients greater than the limiting degree and so exactly $\binom{n}{2}$ independent coefficient multiplications are necessary. Symbolic multiplication of two polynomials of degree n results in a polynomial of degree $2n$, whereas here the result is still degree n, so no algorithm on recombining parts of the polynomial will be faster.

Point–to–point multiplication of polynomials (the Hadamard product) is $\Theta(n)$, similar to the elementary operations.

The operation of computing the kth power of a power series can be accomplished in $\Theta(n^2 \log k)$ by using repeated squaring. We can modify an $\Theta(n^2)$ algorithm by Nijenhuis and Wilf [12] to compute the kth power. Their algorithm computes $g(x) = (1 + f(x))^k$. We can convert any polynomial to such a form in $\Theta(n)$, then after application, convert back appropriately, for an $\Theta(n^2)$ algorithm:

$$
\begin{aligned}
g(x) &= (1+f(x))^k \\
g'(x) &= kf'(x)(1+f(x))^{k-1} \\
&= \frac{kf'(x)(1+f(x))^k}{(1+f(x))} \\
&= kf'(x)g(x) - g'(x)f(x) \\
\sum_{0\le i}(i+1)g_{i+1}x^i &= k\sum_{0\le i}(i+1)f_{i+1}x^i \cdot \sum_{0\le i} g_i x^i - \sum_{0\le i}(i+1)g_{i+1}x^i \cdot \sum_{0\le i} f_i x^i \\
&= \sum_{0\le i}\sum_{0\le j\le i} k(j+1)f_{j+1}g_{i-j}x^i - \sum_{0\le i}\sum_{0\le j\le i}(j+1)g_{j+1}f_{i-j}x^i \\
&= \sum_{0\le i} x^i \sum_{0\le j\le i} k(j+1)f_{j+1}g_{i-j} - (i-j)f_{j+1}g_{i-j} \\
&= \sum_{0\le i} x^i \sum_{0\le j\le i} (k(j+1)-(i-j))f_{j+1}g_{i-j} \\
g_0 &= 1 \\
g_i &= \frac{1}{i}\sum_{1\le j\le i}(kj-i+j)f_j g_{i-j}
\end{aligned}
$$

To get any polynomial into the appropriate form, find the first non–zero coefficient, f_s. Shift all the coefficients left by s and divide by f_s. This makes the f_0 equal 1 and in the correct form. After applying the above algorithm, shift right $k \cdot s$, and multiply by f_s^k.

The rest of the operations are much simpler. The closure of a power series, $\frac{1}{1-f(x)} = \sum_{0 \le i} f(x)^i$ can be computed in $\Theta(n^2)$ by rearranging and solving for coefficients:

$$\begin{aligned}
g(x) &= \frac{1}{1-f(x)} \\
&= 1 + f(x)g(x) \\
\sum_{0 \le i} g_i x^i &= 1 + \sum_{0 \le i} f_i x^i \cdot \sum_{0 \le i} g_i x^i \\
&= 1 + \sum_{0 \le i} \sum_{1 < j \le i} f_j g_{i-j} x^i \\
g_0 &= 1 \\
g_i &= \sum_{1 \le j \le i} f_j g_{i-j}
\end{aligned}$$

Since necessarily $f_0 = 0$ for this to converge, by default $g_0 = 1$ and the rest can be computed in order.

Similarly, the exponential can be computed in $\Theta(n^2)$ by considering the logarithmic derivative of $g(x) = \exp f(x)$ and solving for the coefficients:

$$\begin{aligned}
g(x) &= \exp f(x) \\
[\log g(x)]' &= [\log \exp f(x)]' \\
g'(x)/g(x) &= f'(x) \\
g'(x) &= g(x)f'(x) \\
\sum_{0 \le i} (i+1) g_{i+1} x^i &= \sum_{0 \le i} g_i x^i \cdot \sum_{0 \le i} (i+1) f_{i+1} x^i \\
&= \sum_{0 \le i} \sum_{0 \le j \le i} (j+1) g_{i-j} f_{j+1} x^i \\
g_0 &= 1 \\
g_i &= \frac{1}{i} \sum_{1 \le j \le i} j g_{i-j} f_j
\end{aligned}$$

Since log is the inverse of exp, we can reverse the recurrence relation for exp:

$$\begin{aligned} g_0 &= 0 \\ g_i &= f_i - \frac{1}{i}\sum_{1\le j<i} j g_j f_{i-j} \end{aligned}$$

Instead of using composition of log and $\frac{1}{1-f}$, we can determine the coefficients of $\log\frac{1}{1-f}$ directly for a better constant factor running time:

$$\begin{aligned} g'(x) &= [\log\frac{1}{1-f(x)}]' \\ &= \frac{f'(x)}{1-f(x)} \\ &= f'(x) + g'(x)f(x) \\ \sum_{0\le i}(i+1)g_{i+1}x^i &= \sum_{0\le i}(i+1)f_{i+1}x^i + \sum_{0\le i}(i+1)g_{i+1}x^i \cdot \sum_{0\le i} f_i x^i \\ &= \sum_{0\le i}\left((i+1)f_{i+1}x^i + \sum_{0\le j\le i}(j+1)g_{j+1}f_{i-j}x^i\right) \\ g_0 &= 0 \\ g_i &= f_i + \frac{1}{i}\sum_{1\le j\le i} j g_j f_{i-j} \end{aligned}$$

These take care of all the labeled operations. The unlabeled need them but also some additional computation. By definition, $I_n(A(x)) = A(x)^n$, which we have just seen. For C_n and D_n, a naive algorithm iterates over all divisors of n (for $\Theta(\sqrt{n})$) and multiplies by the totient $\phi(n)$ (which also takes $\Theta(\sqrt{n})$ to compute), for a total $\Theta(n)$ (See Appendix A). For S_n and A_n, we have to sum over all partitions of n. The number of partitions of n is $\mathrm{O}\left(\frac{e^{\sqrt{n}}}{n}\right)$ (see Andrews [1] for a full treatment). Computing the cycle structure and corresponding generating function term is linear in the number of distinct parts of a partition (since that is the form of the partition we compute from). The number of distinct parts is $\mathrm{O}(\sqrt{n})$, so the total time is $\mathrm{O}\left(\frac{e^{\sqrt{n}}}{\sqrt{n}}\right)$. This is better than exponential but worse than $\mathrm{O}\left(n^{\log n}\right)$.

For the full unlabeled families, Cyc and Dih have the same runnign time as C and D. Because Set and Bag simplify so much from S and A, they are not near to exponential. The summation must be handled iteratively but can be sped up by repeated squaring for a running time $\Theta(n^2 \log n)$.

The algorithms and their running times are tabulated as follows:

Operation	Running time
$+, -, 0, 1$	$\Theta(n)$
$\times$	$\Theta(n^2)$
$f(x)^k, \frac{1}{1-f(x)}$	$\Theta(n^2)$
$\exp, \log, \log \frac{1}{1-f(x)}$	$\Theta(n^2)$
I	$\Theta(n^2)$
C, D	$\Theta(n^2 \log n)$
S, A	$O\left(\frac{e^{\sqrt{n}}}{\sqrt{n}}\right)$
Cyc, Dih	$\Theta(n^2 \log n)$
Set, Bag	$\Theta(n^2 \log n)$

Table 3.2: Running time of basic enumerators

3.3.2 Construction from Enumeration

These above algorithms are used to compute the coefficients of generating functions for a family of objects defined by the combinatorial operators. In other words, they enumerate the classes by number of atomic elements they contain. They can also be used to construct the objects in

a family. Bijections between integers and the structures specified by the operations above can be made through the generating functions. Unranking, taking the bijection from integers to structures, is the process that given an integer n, constructs the nth structure of a specification. Given the nature of generating functions, it is convenient to know the size (number of atoms or terminal symbols) in the structure. The general procedure for unranking works recursively by decomposition into previously computed operations. For all these construction algorithms, we first assume that the generating function of the object and its base objects, have already been computed.

Disjoint union can be constructed by considering the two base generating functions: if the object sought of given weight has rank less than the maximum rank of same weight of the first base object, then construct the object from that. Otherwise, subtract the maximum from the rank sought and construct using the second base. So one constructs either from the first or from the second.

Product of two bases uses convolution. A product is an ordered pair, the weights of the two parts adding to the total sought. First, see how many objects there are of weight zero from the first base, and weight n from the second, then weight 1 and weight $n-1$, and so on until the rank sought, r, lies within the appropriate range. For each new range, subtract the (numeric) product of the number of first base objects of size k and the number of second base objects of size $n-k$. Once that range is found, construct the object of weight k from the first base with rank $r \bmod k$, and an object of weight $n-k$ from the second with rank r/k.

A sequence is similar to product, not only in that a sequence is a series of products, but also by comparison of recurrence relations: the second base generating function for sequences is the sequence itself, but always of lower weight.

A set of n elements over a base object can be constructed from its rank. Consider the sum of the number of the base objects up to the size of the set object sought. Use that sum to construct the first element of the set. Subtract ranks and weights accordingly, and continue with the rest of the elements, using ranks less than the one used for the object in the previous set element.

A multiset is constructed almost exactly like a set except the rank of the element object can be less than or equal to that of the previous element.

A cycle is constructed by choosing the first object as for set and multiset but then allowing any element of any rank less than that of the first in the rest of the elements, as though the last $n-1$ elements were a sequence none of whose ranks are greater than the first.

A dihedral is constructed by choosing the first two elements as with multiset, then the rest as with sequence.

3.3.3 Enumerating the Complement

Some combinatorial structures can be defined as the complement of some subset of a set. The best example of this is the free trees (unrooted, unlabeled, no restrictions). It can be defined as the set of rooted trees that are not bisymmetric about an edge. If $T(x)$ counts the rooted trees, then those bisymmetric about an edge is counted by unordered pairs of rooted trees. These are the one–to–one bivalent functions on trees. Removing these from the set of all rooted trees gives Otter's classical formula for free trees:

$$\begin{aligned} t(x) &= T(x) - \mathrm{Set}\,(T(x), \mathrm{card} = 2) \\ &= T(x) - 1/2(T(x)^2 - T(x^2)) \end{aligned}$$

where $T(x) = x \cdot \mathrm{Bag}\, T(x)$, the rooted general trees.

For construction, subtraction of generating functions does not give enough information; it does not tell us exactly which elements of the universe the subset should remove. Assuming we had a ranking function for $t(x)$, given the ith rooted tree, we could check if it is edge symmetric. If so, we would try $i+1$ and so on, until we found one not edge symmetric. This method is not very convenient, since every construction needs a ranking algorithm, and *every* rooted tree needs to be constructed and verified to get the appropriately numbered free tree. This depends on the order of the number of structures of the superset and therefore is usually highly inefficient.

Chapter 4

Enumeration and Construction of Graphs

Graphs, as a class of combinatorial objects, must be handled somewhat specially as compared to what we've already seen; they don't have the convenient recursive substructure that the others have. First, we will analyze them alone, and then in the context of other classes of objects.

Similar to the basic labeled structures above, labeled graphs are fairly simple objects to enumerate and construct (see [8, Chap 1.1]). For an undirected graph on n vertices, there are $\binom{n}{2}$ possible edges, and therefore $2^{\binom{n}{2}}$ graphs. These are then simple to unrank; an ordering of subsets of $\binom{n}{2}$ gives a ranking for the graphs.

Enumeration of unlabeled graphs is not so obvious. Take a random graph on n vertices. Labeled, it is isomorphic to other graphs by some permutation of labels. This preserves the adjacency relation of all vertices, while removing the labels. Indeed, any permutation preserving adjacency is an allowable isomorphism. This naturally leads to the use of groups; namely, the symmetric group of all permutations acting on the n vertices.

4.1 Graph Enumeration by Coset Enumeration

For example, following Sims [16], to enumerate the non-isomorphic graphs of size three, take the set of three vertices labeled 1, 2, and 3. From above, there are $\binom{3}{2} = 3$ edges:

$$\begin{aligned} a &= \{0,1\} \\ b &= \{0,2\} \\ c &= \{1,2\} \end{aligned}$$

and therefore $2^3 = 8$ different labeled graphs:

$$\begin{aligned} G_0 &= \{\} & G_4 &= \{c\} \\ G_1 &= \{a\} & G_5 &= \{a,c\} \\ G_2 &= \{b\} & G_6 &= \{b,c\} \\ G_3 &= \{a,b\} & G_7 &= \{a,b,c\} \end{aligned}$$

The choice of order is by considering the characteristic function for each set of edges as a binary numeral: $c = 2^0, b = 2^1, a = 2^2$. To find out which graphs are isomorphic to each other, that is to determine the partition into sets whose members are isomorphic, we must take all permutations of $\{0,1,2\} = \mathrm{Sym}(3)$ as they act on the graphs. Take, for example, a permutation from Sym(3): $(0\,1)$. It transforms the edges to $\backslash a\, c\, b \backslash$, or equivalently in cycles $(b\, c)$. The set of all these edge permutations will then induce permutations on the graphs. The permutation $(b\, c)$ takes the set $\{a,b\}$ to $\{a,c\}$ or, in terms of the graphs they represent, it takes G_6 to G_5. So for all the graphs, $(b\, c)$ yields the permutation of graphs: $(G_1\, G_2)(G_5\, G_6)$. So if we start with all permutations of nodes and propagate their actions on the set of edges and thereby on the set of graphs, we get a set of permutations on graphs:

$$
\begin{aligned}
() &: () \\
(0\,1\,2) &: (G_1\,G_2\,G_4)(G_3\,G_6\,G_5) \\
(0\,2\,1) &: (G_1\,G_4\,G_2)(G_3\,G_5\,G_6) \\
(0\,1) &: (G_1\,G_2)(G_5\,G_6) \\
(0\,2) &: (G_1\,G_4)(G_3\,G_6) \\
(1\,2) &: (G_2\,G_4)(G_3\,G_5)
\end{aligned}
$$

The graphs in each cycle of one of these permutations are isomorphic. From here it is a simple task to find the overlap of all the cycles in all of these permutations by merging the equivalence classes defined by all the cycles. This final permutation (with fixed points) is $(G_0)(G_1\,G_2\,G_4)(G_3\,G_6\,G_5)(G_7)$. So the final non–isomorphic graphs are, taking a representative from each set, G_0, G_1, G_3, G_7, which are easy to confirm by inspection.

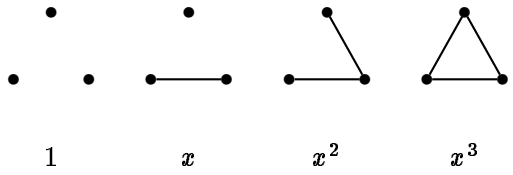

Figure 4.1: The representative non–isomorphic graphs G_0, G_1, G_3, G_7

As to higher order graphs, some of the calculations of permutations can be eliminated by only using the generating permutations for the symmetric group: $(0\,1)$ and $(0\,1\cdots n-1)$. Calculate the images of just these two in the permutations of edges, then calculate the images in the permutations of graphs. Since the two permutations generate $\mathrm{Sym}(n)$, their images in the permutations on graphs will generate all the graph isomorphisms. Instead of trying to calculate all these permutations, all that is needed is to determine the orbits induced by the graph permutations, with an elementary algorithm (see [3]).

For example, to enumerate the non-isomorphic graphs of size four, take the set of four vertices labeled 0, 1, 2, and 3. From above, there are $\binom{4}{2} = 6$ edges:

$$\begin{array}{lllllllll} a & = & \{0,1\} & & b & = & \{0,2\} & & d & = & \{0,3\} \\ & & & & c & = & \{1,2\} & & e & = & \{1,3\} \\ & & & & & & & & f & = & \{2,3\}, \end{array}$$

and therefore $2^6 = 64$ different labeled graphs in order:

$$\begin{array}{lll} G_0 & = & \{\} \\ G_1 & = & \{a\} \\ G_2 & = & \{b\} \\ & \cdots & \\ G_{61} & = & \{a,c,d,e,f\} \\ G_{62} & = & \{b,c,d,e,f\} \\ G_{63} & = & \{a,b,c,d,e,f\} \end{array}$$

The choice of order is by the considering the characteristic function for each set of edges as a (reversed) binary numeral: $a = 2^0, b = 2^1, \cdots f = 2^5$. To find out which ones are isomorphic to each other, that is to determine the partition into sets, we must take all permutations of $\{0,1,2,3\} = \text{Sym}(4)$ as they act on the edges and so acting on graphs. For example, a permutation from Sym(4), (0 3 1), transforms the edges to $\backslash c\ f\ e\ b\ a\ d \backslash$, or equivalently the permutation $(a\,c\,e)(b\,f\,d)$. The set of all these edge permutations will then induce permutations on the graphs. For example, with a little computation, the permutation $(a\,c\,e)(b\,f\,d)$ takes $G_{28} = b, c, d$ to $G_{37} = f, e, b$. The full transformation will induce a permutation of graphs to graphs, whose equivalence classes of the permutations are shown in Table 4.1 and canonical members of each equivalence class in Figure 4.2.

(0)	no edges
(1 2 4 8 16 32)	1 edge
(33 12 18)	2 unconnected edges
(3 5 6 9 10 17 20 24 34 36 40 48)	2 connected edges
(11 21 44 56)	'crow's foot' $= K_{1,3}$
(52 42 19 7)	C_3
(13 14 22 25 26 28 35 37 38 41 50 59)	P_4
(30 51 45)	C_4
(15 23 27 29 39 43 46 53 54 57 58 60)	'oil can' $= C_3 + e$
(31 47 55 49 61 62)	5 edges $= K_4 - e$
(63)	6 edges $= K_4$

Table 4.1: Equivalence classes of graphs induced by permutations on nodes

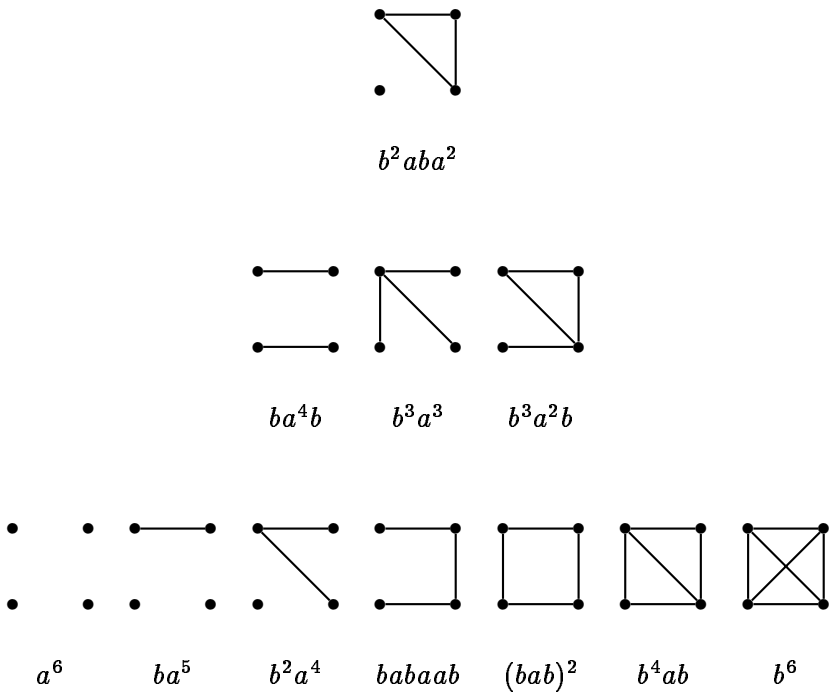

Figure 4.2: The non–isomorphic graphs on four points

This derivation both enumerates and generates in order the unlabeled graphs, first by counting the number of orbits, and second by using the characteristic vector (the most efficient representation for doing the above permutation calculations) of the representative of each orbit. However, because we must work on the entire set of labeled graphs, the space complexity is exponential $\mathrm{O}\left(2^{n^2}\right)$ and the time complexity at least that much. Also, in contrast to the methods at the beginning, this method does not lend itself well to generalization or specialization.

4.2 Graph Enumeration by Pólya Enumeration

A classical result of Harary's (known but not published by Pólya), is the derivation of a generating function for enumerating graphs using the cycle index for S_n. The cycle index for all permutations of n elements is given in equation 3.9. These permutations induce permutations

on the edges as described above. Consider a permutation on $[n]$; it induces a permutation on the group of $[\binom{n}{2}]$ pairs on $[n]$. These pairs can be considered as edges of an undirected graph. The cycle index for this "pair group" is called $S_n^{(2)}$. This is a cycle index polynomial with the same coefficients as S_n but on terms that are representative of the permutations on graph edges, corresponding to the permutations on graph nodes; the monomial for the edge permutations is a function of the monomial for the node permutations.

To compute the term in $S_n^{(2)}$ from S_n, there are two cases. First, the pair may come from two different cycles in the permutation from S_n. If the two point cycles are of length i and j, then there will be $\gcd(i,j)$ edge cycles of length $\text{lcm}(i,j)$ (abbreviated (i,j) and $[i,j]$ respectively). Since there are e_i and e_j point cycles, the corresponding factor is $x_{[i,j]}^{(i,j)e_ie_j}$. If $i = j$ this over counts the cycles; we only have a factor $x_k^{k\binom{e_k}{2}}$. For example, in Figure 4.3, the point cycles (0 1)(2 3 4 5) induce the edge cycles (a b c d)(e f g h) with $\gcd(2,4) = 2$ cycles of length $\text{lcm}(2,4) = 4$.

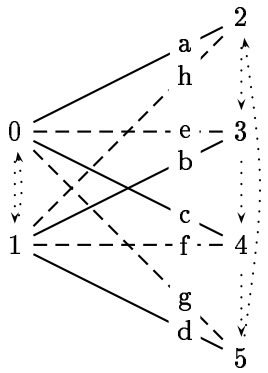

Figure 4.3: Disjoint point cycles inducing cycles on edges between

Second, the pair may come from within a single cycle of length k. If k is odd $(k = 2i+1)$, there will be i edge cycles of length $2i+1$ for a factor of $x_{2i+1}^{ie_{2i+1}}$. For example, in Figure 4.4,

(0 1 2 3 4) induces the edge cycles (a c f j g)(b e i d h) with $i = 2$ cycles of length $2i + 1 = 5$. Note that the labels are in order based on the order of the nodes; this somewhat obtuse edge ordering allows adding a node without changing the labels on the edges.

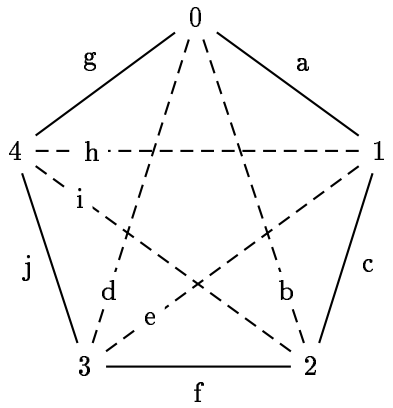

Figure 4.4: An odd point cycle and the induced edge cycles

If k is even $(k = 2i)$, there will be $i - 1$ cycles of length $2i$ and one cycle of length i [for a total factor of $(x_i x_{2i}^{i-1})^{e_{2i}}$] because an even cycle will induce a single half length cycle on edges from point i to point $2i$. For example, in Figure 4.5, the point cycle (0 1 2 3 4 5) induces the edge cycles (a c f j o k)(b e i n j l)(d h m) with $i - 1 = 2$ cycles of length $2i = 6$, and one cycle of length $i = 3$.

So, for the complete equation, over all cases:

$$S_n^{(2)} = \sum_{e \in \lambda(n)} \left(\prod_k \frac{1}{k^{e_k} e_k!} \cdot \prod_{k=2i+1} x_{2i+1}^{i e_{2i+1}} \cdot \prod_{k=2i} (x_i x_{2i}^{i-1})^{e_{2i}} \cdot \prod_k x_k^{k\binom{e_k}{2}} \cdot \prod_{i<j} x_{[i,j]}^{(i,j) e_i e_j} \right)$$

For example, the pair group $S_4^{(2)}$ can be easily computed from S_4 as follows:

$$S_4 = \frac{1}{24}(x_1^4 + 8x_3x_1 + 3x_2^2 + 6x_2x_1^2 + 6x_4)$$

yields

$$S_4^{(2)} = \frac{1}{24}(x_1^6 + 8x_3^2 + 3x_2^2x_1^2 + 6x_2^2x_1^2 + 6x_4x_2)$$

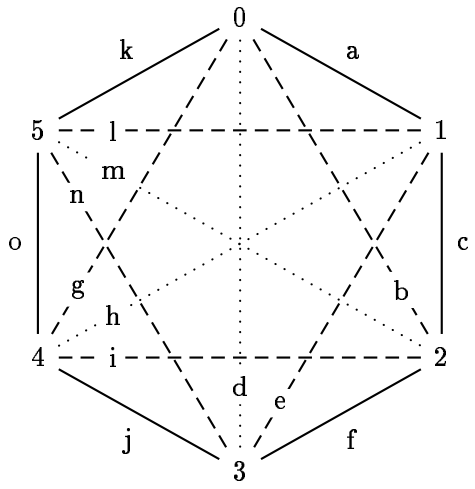

Figure 4.5: An even point cycle and the induced edge cycles

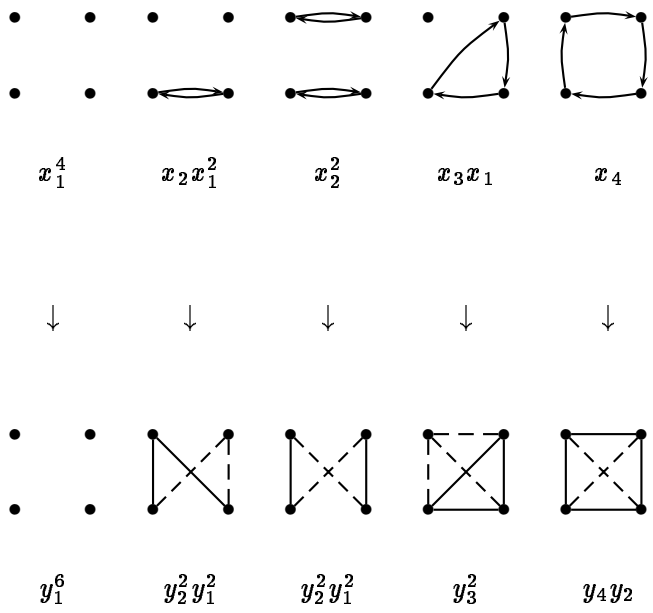

Figure 4.6: The mapping of point cycles to edge cycles for S_4

as depicted in Figure 4.6.

For directed graphs, we consider the group on (restricted) ordered pairs, called $S_n^{[2]}$. Again, since this is induced by permutations on the points, we look at the generic cycle structure of integer partitions. If the directed edges are between two different cycles of length i and j, the factor involves $x_{[i,j]}^{(i,j)e_ie_j}$ from the unordered case, but is squared to account for both directions. When $i = j$, the factor is similarly formed: $x_k^{2k\binom{e_k}{2}}$. If the points are in the same cycle, there are $2\binom{k}{2} = k(k-1)$ directed edges within the cycle, which yields $k-1$ cycles of length k, same as in the case for odd cycles on unordered pairs. This also accounts for cycles of even length here because a directed edge will not match itself after halfway through the cycling as an unordered edge would. So the complete equation is:

$$S_n^{[2]} = \sum_{e\in\lambda(n)} \left(\prod_k \frac{1}{k^{e_k} e_k!} \cdot \prod_k x_k^{(k-1)e_k} \cdot \prod_k x_k^{2k\binom{e_k}{2}} \cdot \prod_{i<j} x_{[i,j]}^{2(i,j)e_ie_j} \right)$$

To account for loops (self–directed edges), an extra factor $\prod_k x_k^{e_k}$ is needed. This is then the cycle index for the unrestricted ordered pair group, denoted S_n^2.

The cycle index generating functions for families of graphs and digraphs are as follows with some simplification of exponents (all product ranges are from 1 to n):

$$\sum_{n\geq 1} S_n^{(2)} = \sum_{n\geq 1} \sum_{e\in\lambda(n)} \left(\prod_k \frac{1}{k^{e_k} e_k!} \cdot \prod_{k=2i+1} x_{2i+1}^{ie_{2i+1}} \cdot \prod_{k=2i} (x_i x_{2i}^{i-1})^{e_{2i}} \cdot \prod_k x_k^{k\binom{e_k}{2}} \cdot \prod_{i<j} x_{[i,j]}^{(i,j)e_ie_j} \right)$$

$$\sum_{n\geq 1} S_n^{[2]} = \sum_{n\geq 1} \sum_{e\in\lambda(n)} \left(\prod_k \frac{1}{k^{e_k} e_k!} \cdot \prod_k x_k^{ke_k^2 - e_k} \cdot \prod_{i<j} x_{[i,j]}^{2(i,j)e_ie_j} \right)$$

$$\sum_{n\geq 1} S_n^{2} = \sum_{n\geq 1} \sum_{e\in\lambda(n)} \left(\prod_k \frac{1}{k^{e_k} e_k!} \cdot \prod_k x_k^{ke_k^2} \cdot \prod_{i<j} x_{[i,j]}^{2(i,j)e_ie_j} \right)$$

So now that we have a generating function for these different types of graphs, what does it mean when we apply it to another generating function? Since the transformation of the

permutation group takes points into edges, the generating functions act on edges. Any function to which one of these graph enumerators is applied will place objects on the graph edges. For example, let the function be $1 + x$. The 'x' will produce an edge in the graph, the '1' will not. So generating function $S_n^{(2)}[1+x]$ will enumerate the non–isomorphic graphs on n vertices. After the manipulation of polynomials and their coefficients, the non–isomorphic graphs on 4 points are counted by:

$$S_4^{(2)}[1+x] = 1 + x + 2x^2 + 3x^3 + 2x^4 + x^5 + x^6$$

where the coefficient of x^i is the number of graphs on 4 points with i edges. These coefficients can be seen in the example in Figure 4.2.

The running time on these graph enumeration algorithms starts with the number of partitions which is $\mathrm{O}\left(\frac{e^{\sqrt{n}}}{n}\right)$. For each partition, computing the part of the term corresponding to induced edge cycles within an odd or even cycle is $\mathrm{O}\left(\sqrt{n}\right)$ (the maximum number of distinct parts). Between cycles is $\mathrm{O}\left(\sqrt{n}^2\right) = \mathrm{O}\left(n\right)$ pairs with $\mathrm{O}\left(\log n\right)$ for the gcd/lcm computation on cycle lengths $\leq n/2$. The total running time is then $\mathrm{O}\left(e^{\sqrt{n}} \log n\right)$.

Multigraphs are graphs with unlimited edges between points or equivalently graphs with weighted edges counted by

$$S_n^{(2)}[1 + x + x^2 + \ldots] = S_n^{(2)}\left[\frac{1}{1-x}\right] = S_n^{(2)}[\mathrm{Seq}(x)].$$

Graphs with colored edges can be determined by

$$S_n^{(2)}[x + y + z + \ldots]$$

where the variables stand for the different colors.

Relations with given properties can be counted (as in Davis [5]). Unrestricted relations on n objects are counted by $S_n^2[1 + x]$, the unrestricted directed graphs. Irreflexive relations

(or equivalently reflexive) are counted by $S_n^{(2)}[1+x]$, the directed graphs with no self–loops. Symmetric, irreflexive relations are counted by $S_n^{[2]}[1+x]$.

Connected graphs (directed or undirected) can be enumerated by considering that any graph is a multiset of its components. This means if $\mathrm{UGrf}(A(x))$ is the generator for graphs with edges from $A(x)$, then the number of connected graphs $\mathrm{Conn}(A(x))$ is specified by:

$$\mathrm{Grf}(A(x)) = \mathrm{Bag}(\mathrm{Conn}(A(x))).$$

In the labeled case (Bag being equivalent to Set), taking logs gets us:

$$\mathrm{Conn}(A(x)) = \log(\mathrm{Grf}(A(x)))$$

whose coefficients are computable using our previously defined log operation. For unlabeled graphs, we have from the definition for Bag:

$$\sum_{1\leq i} \frac{\mathrm{Conn}(A(x^i))}{i} = \log(\mathrm{Grf}(A(x)))$$

By inspecting the coeffecients, we find that we can apply classical Möbius inversion to get the coefficients for Conn:

$$
\begin{aligned}
G'(x) &= \log(\mathrm{Grf}(A(x))) \\
&= \sum_{1\le i} \frac{\mathrm{Conn}(A(x^i))}{i} \\
&= \begin{array}{ccccccccccccc}
c_1 & + & c_2 & + & c_3 & + & c_4 & + & c_5 & + & c_6 \ldots \\
 & + & c_1/2 & + & & + & c_2/2 & + & & + & c_3/2 \ldots \\
 & & & + & c_1/3 & & & & & + & c_2/3 \ldots \\
 & & & & & + & c_1/4 & & & & \ldots \\
 & & & & & & & + & c_1/5 & & \ldots \\
 & & & & & & & & & + & c_1/6 \ldots
\end{array} \\
&= \sum_{1\le i}\sum_{j|i} \frac{c_{i/j}x^i}{j} \qquad \text{summing by columns} \\
\mathrm{Conn}(A(x)) &= \sum_{1\le i}\sum_{j|i} \frac{\mu(j)}{j} g_{i/j}x^i \qquad \text{by Möbius inversion} \qquad (4.1)
\end{aligned}
$$

This calculates the coefficients for the inverse of the multiset operation. The running time of the calculation using 4.1 is $\mathrm{O}(n)\cdot\mathrm{O}(\tau)\cdot\mathrm{O}(\mu)$. The last two items are discussed in Appendix A and are both $\Theta(\sqrt{n})$. From these results, the running time for computing connected components is $\mathrm{O}(n^2)$

The operator Conn is most obviously used for counting connected subclasses of species of graphs. One might wonder what the same multiset inverse applied to the other constructions means combinatorially. Those whose top level constructor is Set or Bag are easy to analyze; for example, the "connected" set partitions ($\mathrm{SP} = \mathrm{Set}(\mathrm{Set}(z, \mathrm{card} \ge 1))$) are combinatorially those with a single partition, of which there is only one for each n (and none for zero), as can be seen directly from the application of log to the generating function.

A more interesting example would not have head term exp or $\times$. It certainly can't be Cyc or Dih because the generating function produced by log has zero for its first coefficient, and applying log to this will not converge. So all that is left is Seq. This immediately gives Cyc in the labeled case; the connected sequences *are* the cycles. For unlabeled structures, the correspondence is not immediate. By the generating function, it is the first term in the summation that computes Cyc. Combinatorially, it is those sequences of a single cycle. The connected Bags are the base objects with identical element sequences replacing each atom. Similarly, the connected Sets are the one–to–one functions on the base set with identical element sequences replacing each atom.

The running times for the graph and connected constructions are in Table 4.2.

Operation	running time
graph	$\mathrm{O}\left(e^{\sqrt{n}}\log n\right)$
connected	$\mathrm{O}\left(n^2\right)$

Table 4.2: Running time of graph and connected operators

Chapter 5

Power Series, Languages, and Semirings

5.1 Definition

The connection between power series and languages began with Chomsky and Schützenberger [4]; they note the relation between the words in a language and formal power series. Kuich and Salomaa [10] describe rational and algebraic power series and their respective relation to regular and context–free languages.

The transition from the polynomial enumerating function for a class to the construction of grammars for the class is possible because of their similar algebraic structure: they are both semirings. A semiring is a closed algebraic system $<R, +, \times, 0, 1>$ where R is the set of elements, $+$ and $\times$ binary operations on R, with elements $0, 1 \in R$ (not necessarily distinct) satisfying the following properties:

- $(R, +, 0)$ is a commutative monoid: $+$ is associative and commutative, with 0 the identity.

- $(R, \times, 1)$ is a monoid.: $\times$ is associative with 1 the identity.

- $\times$ distributes over $+$ from the left and right: $a \times (b+c) = a \times b + a \times c$, $(a+b) \times c = a \times c + b \times c$

Rings are semirings with additive inverse, i.e. a ring is a semiring with addition augmented from a monoid to a group. Notice that under this axiomatization, $\cdot$ is not necessarily commutative, and 0 may not be an annihilator ($a \cdot 0$ and $0 \cdot a$ do not necessarily equal 0). We can define matrices over a semiring, where

$$[A +_{\mathrm{M}} B]_{ij} = [A]_{ij} + [B]_{ij}$$

and

$$[A \cdot_{\mathrm{M}} B]_{ij} = \sum_k [A]_{ik} \cdot [B]_{kj}$$

This definition satisfies all the semiring axioms itself, except that there is no multiplicative identity, which is not necessary.

Both formal power series and languages are semirings. Formal power series over $\mathbf{N}$ (for enumeration) form a semiring. Languages form a non–commutative semiring in the following manner (following Kuich and Salomaa [10]): Let S be the set of sets of tuples over a finite alphabet, $+$ is disjoint union of sets of tuples, with $0 = \emptyset$ the empty set, $\times$ is concatenation of sets of tuples, where concatenation of two tuples is the first followed by the second, concatenation of two sets is the set of all possible ordered pairs concatenated, and $1 = \{\epsilon\}$ where ϵ is the empty string of length zero. Distributivity follows from the definition of union and concatenation.

A regular language has three equivalent characterizations: it is a language that is:

- accepted by a finite automaton,

- recognized by a grammar whose productions are from $V \to (V + 1)T$,

- generated by an expression using $+$, $\cdot$, and * (the latter called closure, defined as $a^* = 1 + a \cdot a^*$).

Because of the latter characterization, any combinatorial structure formed using union, product, and sequence can now be specified as a regular language (with the restriction imposed by the second characterization).

We can define a particular regular language by a system of equations over words and variables:

$$\begin{aligned} x_1 &= A_{11}x_1 + A_{12}x_2 + \cdots + A_{1n}x_n + b_1 \\ x_2 &= A_{21}x_1 + A_{22}x_2 + \cdots + A_{2n}x_n + b_2 \\ &\vdots \qquad\qquad\qquad\qquad \vdots \\ x_n &= A_{n1}x_1 + A_{n2}x_2 + \cdots + A_{nn}x_n + b_n \end{aligned}$$

where the x_i are unknown sets of strings defined by the mutually recursive equivalences, $A_{ij} = \sum t, t \in T$ such that there is a rule $x_i \rightarrow x_j t$, and the b_i are finite sets of strings of length one corresponding to the rule $x_i \rightarrow t, t \in T$. Note that all A_{ij} and b_i are constants in the semiring of strings. By convention, x_1 is the principle language being defined by the system. This encodes a grammar for a regular language as a linear system of equations with variables (the unknown sets being solved for), and the constants (the finite sets of words)[1].

In this light, we can view the system vectors and matrices over the semiring elements:

$$\vec{x} = A \cdot \vec{x} + \vec{b}$$

The solution to this equation is $\vec{x} = A^* b$, where A^* is the closure of the matrix over a semiring, with appropriate use of the semiring operations, where $A^* = I + A \cdot A^*$. Lehmann

[1] The equivalence with Deterministic Finite Automata can easily be seen if we consider the variables x_i as the states, x_1 as the start state, a transition from state x_j to x_i with edge label $A_{ij} = t$ if $x_i \rightarrow x_j t$, and b_i is either null if x_i is not a final state or ϵ if it is

[11] showed that Gaussian elimination on systems of linear equations over semirings can solve this system in $O(n^3)$. This algorithm is identical to Kleene's algorithm for closure over regular languages. The solution of such a system is in terms of the constants using the operations $+$, $\cdot$, and *.

The polynomial interpretation of such a solution is in terms of enumerating generating functions, mapping the semiring of a regular language to the semiring of multivariate polynomials over the naturals whose variables are the alphabet of the language. The coefficient of a term $x_1^{p_1} \cdot x_2^{p_2} \ldots x_i^{p_i}$ is the number of words in the specified language that have p_i letters x_i.

5.1.1 Context–free Grammars

Most of the structure we have looked at have recurrent grammars that are not regular. For example, rooted plane trees, with definition $\mathrm{RPT} = z \cdot \mathrm{Seq}(\mathrm{RPT}) \equiv \mathrm{RPT} = z \cdot T,\ T = 1 + \mathrm{RPT} \cdot T$, is not regular because of the non–regular recursive Seq (which gives the concatenation of two variables). If we consider context–free grammars (CFGs), many more of our structures can be encoded. Without loss of generality, we can assume that our grammar is in Chomsky normal form (CNF) where all productions are from $V \to (T + V \cdot V)$.[2] Grammars in this form can also be represented as a matrix equation:

$$\vec{x} = A \cdot \vec{x} + \vec{b}$$

where the x_i are the sets being defined, A_{ij} a constant set of strings $A_{ij} = \sum x_k, x_k \in V$ such that there is a rule $x_i \to x_j x_k$, and the b_i are finite sets of strings of length one corresponding to the rule $x_i \to t, t \in T$. Note that only the b_i are constants in the semiring.

[2] Note the notational ambiguity: the class of CFGs is be specified by a regular language.

Following Salomaa and Soittola [14], this system of equations also has a solution by closure. It is not necessarily terminating as for regular languages, but it is convergent and so can be computed up to an arbitrary precision, i.e. to an arbitrary word length. Here again, a polynomial interpretation is just a homomorphism of letters to variables, the coefficients being positive integers that count the number of derivations.

Chapter 6

Conclusion

6.1 Previous Work and Implementations

Most other work in construction of combinatorial objects (Reingold, Nievergelt, and Deo [13] and Nijenhuis and Wilf [12]) concentrates on the optimal algorithms for sequential construction of permutations, gray codes, and integer partitions. Joyal [9] gave a general method for constructions, and both Bergeron [2] and Zimmermann [18] expanded upon them.

Several systems have been developed: Nijenhuis and Wilf [12] in Fortran, Skiena [17] in Mathematica, and Bergeron [2]. Paul Zimmermann [18] has written a Maple package for random generation according to the grammar of construction which in effect gives algorithms for ordered generation. His work forms the starting point for the work reported here.

The above operations for enumeration of combinatorial objects ($+$, $-$, $\times$, Seq, Cyc, Dih, Set, Bag, I_n, C_n, D_n, A_n, S_n, $S^{[2]}$, $S^{(2)}$, S^2, Conn, UGrf, DSGrf, and DGrf) and for construction ($+$, $-$, $\times$, Seq) have been implemented as a library of C++ classes and is available by contacting the author at e–mail address `maharri@cs.uiuc.edu`.

6.2 Uses

Sequential construction of combinatorial objects has many uses. Many algorithms act over all possible arrangements or configurations. Listing combinations, n-tuples, partitions, words over a language: these are all common subroutines needed in systems. The construction algorithms make all these combinations easy to implement.

Given a particular class of combinatorial objects, we often want to submit them to new algorithms to test their efficiency. With full enumeration and construction, we can produce data for the whole distribution or from a uniform sampling. This will allow one to make conjectures about properties of an algorithm.

Experimental mathematics or the use of natural scientific methods to aid in discovering mathematical knowledge is becoming more feasible with better hardware. The algorithms above will allow perfect knowledge of smaller sets and probabilistic knowledge for sets intractably large. Of course, this makes no replacement for the discovery of proofs of properties of these sets, only it may make conjectures easier to corroborate and thereby give more hope to the search for proof.

Unranking is essentially a function from $\mathbf{N}$ to a set of objects. The object corresponding to a particular number can then be thought of as an encoding of the number. The traditional digital system encodes numbers as n–tuples of ten distinct atoms. All the bijections support old, and give new, numbering schemes. Factorials and Fibonacci numbers are two classical examples of sequential encodings of numbers.

6.3 Future Work

Many combinatorial objects have been left out of the above catalog. Young tableaux, block designs and Latin squares, groups, orders, and other algebras, and many various extensions and restrictions of previously mentioned objects. Either suitable constructions for these objects out of the above operations may be discovered, or new operations for them may be formulated that will fit into this framework.

Nijenhuis and Wilf [12] considered the set of useful tasks in combinatorial construction on an ordered set of objects to be:

- ranking - given an object in the set, determine the integer (the rank) specifying that object.
- unranking - given an integer, construct the corresponding object.
- sequencing - given an object, construct the 'next' object in the order.
- randomization - select an object uniformly at random from the set.

Unranking was the subject here. Given an unranking algorithm for a set, randomization can be thought of as a trivial extension by using a random generator of integers over the number of items in the set.

Ranking is not necessarily a simple reversal of the unranking algorithms. If we can assume that an object presented to be ranked has an identical encoding as the object constructed with the corresponding rank, then such an inverse operation would not be difficult to formulate. Barring that assumption, which is most likely false in real applications, there needs to be preprocessing for the objects to normalize them to a form appropriate to the particular class to which it belongs. This normalization is certainly nontrivial.

Sequencing could be considered trivial: for an object x, Unrank(Rank(x)+1) suffices. Often this is all we want to do. However for constructing large sets in sequence, we may be able to speed up construction of the whole set with smaller amortized running time for single objects by having a clever 'next' operation. The canonical example might be the simplest: the construction of the binary representation of integers. The canonical unranking algorithm takes time $\Theta(\log n)$ for each number in the set $< n$. for a total $\Theta(n \log n)$. By incrementing in a ripple–carry counter, the amortized cost is $\Theta(n)$. Efficient next algorithms could be made for all the above objects. Nijenhuis and Wilf [12] give a number of examples for this, notably the Trotter algorithm for permutations. For an exhaustive survey of permutation algorithms, see Sedgewick [15].

The following are a number of unsolved problems posed here which would be desirable to solve:

- Converting an enumerator for an arbitrary permutation group and operations on such generating functions to efficient constructors.
- Efficient computation of cycle indices for specific classes of groups or operations on groups.
- Solving the graph isomorphism problem. This means finding the polynomial equivalence class to which it belongs. If it is found to be a proper subset, which it appears to be, then such a solution would show that P $\neq$ NP.
- Determining a combinatorial interpretation of unrestricted grammars, and solutions of systems in terms of generating functions and constructions.
- Finding grammatical rules corresponding to the cycle and set constructions.

- Enumerating and constructing classes with other definitions. Transitive relations (posets, lattices, etc.) and non–isomorphic algebras. Finding a general method for complement construction.

The main results achieved are the derivations of the generating functions of the basic combinatorial operators through Pólya enumeration, a general method for using generating functions as constructors, an algorithm for constructing graphs, an algebraic interpretation of generating functions as semirings, and a solution of systems of combinatorial equations.

Appendix A

Running Time for Number Theoretic Functions

The following number theoretic functions

- $\phi(n)$ = the number of positive integers less than (or equal) and relatively prime to n,
- $\tau(n)$ = the number of positive divisors of n,
- $\mu(n) = 1$ if n has an even number of unique prime divisors, -1 if odd, and 0 otherwise (n is not square–free, i.e. has multiple prime factors),

are all multiplicative, that is, for f multiplicative, $f(a \cdot b) = f(a) \cdot f(b)$. This leads to a simple method for computation for such functions based on the prime factorization: if $n = p_1^{r_1} p_2^{r_2} \ldots p_k^{r_k}$, then:

$$f(n) = \prod_{i=1}^{n} f(p_i^{r_i})$$

For each of these functions there is a simple method for calculating $f(p^r), p$ a prime:

$$\begin{aligned}
\phi(p^r) &= (p-1)p^{r-1} \\
\tau(p^r) &= 1+r \\
\mu(p^r) &= 1(r=0), -1(r=1), 0(r>1)
\end{aligned}$$

For arbitrary n, these can all be computed in time linear in the number of factors, given such a factorization. The number of factors is $\mathrm{O}(\log n)$. To factor, we use a naive trial division algorithm, attempting 2 through $\sqrt{n}$, for an $\Theta\left(n^{1/2}\right)$ run time.

The size of the values from these functions is in Table A.1

function	computation	value
ϕ	$\Theta\left(n^{1/2}\right)$	$\mathrm{O}(n)$
τ	$\Theta\left(n^{1/2}\right)$	$\mathrm{O}(\log n)$
μ	$\Theta\left(n^{1/2}\right)$	$\Theta(1)$

Table A.1: Running time of number theory functions

Any algorithm that uses $\sum_{j|i}$ iterating over all divisors has a factor of $\mathrm{O}(\tau) = \Theta\left(n^{1/2}\right)$ in its running time.

Note that it is commonly understood that integer factorization has exponential running time, contrary to what we claim here. Running time is calculated according to the size of the input. With integer factorization (and other operations like addition and multiplication) the input size is the length of the bit encoding, which is $\log n$ of the value. Turned around, the value is exponential in the length of the encoding, yielding the 'slowness' of factorization. However, we are not factoring the coefficients of these enumerating polynomials; we factor the indices of

the polynomial elements. So we are taking the input size interpretation as being the length of the polynomial which is the maximum degree for which a coefficient is calculated.

Bibliography

[1] G. E. Andrews. *The Theory of Partitions.* Addison-Wesley, Reading, MA, 1976.

[2] F. Bergeron. Algorithms for the sequential generation of combinatorial structures. *Discrete and Applied Mathematics*, 24(1–3):29–35, 1989.

[3] G. Butler. *Fundamental Algorithms for Permutation Groups.* Number 559 in Lecture Notes in Computer Science. Springer-Verlag, Berlin, 1991.

[4] N. Chomsky and M. Schützenberger. *The Algebraic Theory of Context-Free Languages*, pages 118–161. North-Holland, Amsterdam, 1963.

[5] R. L. Davis. The number of structures of finite relations. *Proceedings of the American Mathematical Society*, 4:486–495, 1953.

[6] P. Flajolet, B. Salvy, and P. Zimmermann. Automatic average-case analysis of algorithms. Technical Report 1233, INRIA, Rocquencourt, August 1990.

[7] L. A. Goldberg. Automating Pólya theory: The computational complexity of the cycle index polynomial. *Information and Computation*, 105:268–288, 1993.

[8] F. Harary and E. M. Palmer. *Graphical Enumeration.* Academic Press, New York, 1973.

[9] A. Joyal. Une theorie combinatoire des séries formelles. *Advances in Mathematics*, 42(1):1–82, 1981.

[10] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5 of *EATCS Monographs in Computer Science*. Springer-Verlag, Berlin, 1986.

[11] D. J. Lehmann. Algebraic structures for transitive closure. *Theoretical Computer Science*, 4:59–76, 1977.

[12] A. Nijenhuis and H. S. Wilf. *Combinatorial Algorithms*. Academic Press, New York, 1978.

[13] E. M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial Algorithms: Theory and Practice*. Prentice-Hall, Englewood CLiffs, NJ, 1977.

[14] A. Salomaa and M. Soittola. *Automata-Theoretic Aspects of Formal Power Series*. Springer-Verlag, New York, 1978.

[15] R. Sedgewick. Permutation generation methods. *Computing Surveys*, 9:137–164, 1977.

[16] C. C. Sims. *Abstract Algebra: A Computational Approach*. John Wiley and Sons, Inc., New York, 1984.

[17] S. Skiena. *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*. Addison-Wesley, Reading, MA, 1990.

[18] P. Zimmermann. Gaïa: A package for the random generation of combinatorial structures. *MapleTech*, 1(1):38–46, 1994.