ORDERED CONSTRUCTION OF COMBINATORIAL OBJECTS

MITCH HARRIS AND NACHUM DERSHOWITZ*

Abstract. The generating function method for counting species of combinatorial objects is applied to the construction of the objects in order. The species considered are those described using context-free grammars with additional group-invariant operators. Some species constructible with this method are integer partitions, rooted trees of specified or unbounded degree including binary and ordered trees, and derivations of words in a context free language. Bijections, algorithms, and decidability results are presented.

Key words. enumeration, construction, unranking, species, generating functions, semiring

AMS subject classifications. 05A15, 68Q50, 16Y60

1. Introduction.

The use of generating functions is a traditional method of counting combinatorial objects; the coefficients of a particular polynomial term correspond to the number of objects of size equal to the exponent, manipulations of the polynomial (formal power series) correspond to combinatorial operations on the set of objects. By an appropriate interpretation, we can use polynomial operations to describe not only the number, but also the combinatorial construction of the set of objects. For example, the subsets of a set of size n are enumerated by the generating function $f(x) = (1+x)^n$, where the coefficient of x^k in $f(x)$ equals $\binom{n}{k}$, and this corresponds to those subsets, those combinatorial objects, with exactly k elements. The polynomial $(1+x)^n$ can be viewed as a sequence of length n, an n-tuple, of choices of either a 1 or an x, where the number of sequences with k x's (k is the weight of the object) is the coefficient of x^k.

The species (the sets of objects) that we will consider are unlabeled objects described by a context free grammar plus the operations of invariance under certain permutation groups. This will provide us with a correspondence between words (derivations) and objects on the one hand, and languages and species on the other.

Flajolet and Zimmermann develop similar techniques [5, 15] but without Polyá enumeration. Related research includes: Alonso and Schott [1] who consider fast random generation techniques, Bergeron et al. [3] who give an extensive algebraic treatment of species, and Jerrum, Vazirani, and Valiant [8] who give an abstract analysis of the complexity of generation. Hickey and Cohen [7] and Mairson [11] give algorithms for constructing words from unambiguous context–free grammars. The concerns here are for giving an efficient, constructive, bijective method (ranking/unranking) for a general class of species.

The plan of the paper is as follows. First, we will show the algebraic correspondence between an encoding of a species and its enumerating generating function. Second, we will show how some species can be described by a system of equations. And third, we will show how the constructions are computed and how efficiently.

2. Semirings: Languages and Power Series.

We consider a set of combinatorial objects to be encoded as a language, a set of words over some alphabet. We define the traditional operation 'concatenation' or '$\cdot$' (sometimes expressed using juxtaposition instead of the operator) on words, which is a monoid with identity ϵ, the unique zero length string. The concatenation of two languages $A \cdot B$ is the set

* Department of Computer Science, University of Illinois/Urbana-Champaign, Urbana, IL 61801, (maharri,nachum@cs.uiuc.edu).

$\{a \cdot b : a \in A, b \in B\}$; this also forms a monoid with identity the set $\{\epsilon\}$. We will also use the disjoint set union operation $A + B$; it is a non-idempotent $(a \neq a + a)$ commutative monoid with identity the empty set, $\emptyset$.

An enumerating generating function for a species is a formal power series, a multivariate polynomial with coefficients from $\mathbf{N}$. The meaning of the polynomial is that the coefficient $k_{i,j\ldots}$ of the monomial $x^i y^j \ldots$ is the number of objects in the species whose encoding has i symbols corresponding to x, j to y, and so on. Addition and multiplication of polynomials is traditional. This interpretation gives the obvious homomorphism from the species to a generating function, where the latter is lacking all information about sequence in the encoding.

Each is a semiring, that is an algebra $<S, +, 0, \cdot, 1>$ where $<S, +, 0>$ is a commutative monoid, $<S, \cdot, 1>$ is a monoid, and $\cdot$ distributes over $+$ from both the left and right, i.e. $a(b + c) = ab + ac, (a + b)c = ac + bc$. Both semirings allow a closure operation, a^*, satisfying the identity $a^* = 1 + a \cdot a^*$ for $a \neq 1$ (see Lehmann [10] and Kuich and Salomaa [9]) which for languages is Kleene closure and for polynomials is $(1 - a)^{-1}$.

The correspondence between polynomials and languages goes back to Schützenberger and Chomsky [4]. We make explicit here the use of polynomial operations to create a bijection between the generating function and combinatorial constructions and also to use that bijection to construct words from the language (unranking) and uniquely identify words (ranking).

3. The Correspondence. What in essence we want to do to get a bijection for constructible species is to give a bijection with the natural numbers. We could take any combinatorial description of the naturals; for simplicity's sake we choose the species with a single atom, s, generated freely (i.e. $\epsilon \in \mathbf{N}$ and if $a \in \mathbf{N}$, then $s(a) \in \mathbf{N}$), so that there is exactly one word of length n for each natural number n:

$$\mathbf{N} = \epsilon, s, ss, sss, ssss, \ldots$$

where ϵ is the zero-length string. The atom or letter s in the language corresponds to the variable s in the generating function:

$$\mathbf{N}(s) = 1 + s + s^2 + \ldots = \frac{1}{1 - s}$$

which counts the number of encodings for each natural number.

Let A, B be two species with corresponding generating functions $A(x), B(x)$.

Suppose we have a countable set A of size a. For the moment we don't care what the structure of each member of the set is. There always exists a bijection between A and some subset of $\mathbf{N}$, so we can give A the following generating function for its encoding by $\mathbf{N}$:

$$\begin{aligned} A(s) &= 1 + s + s^2 + \ldots \\ &= \sum_{0 \leq k < a} s^k \\ &= \begin{cases} \dfrac{s^a - 1}{s - 1} & \text{if a is finite,} \\ \dfrac{1}{1 - s} & \text{otherwise.} \end{cases} \end{aligned}$$

If we can convert the generating function for a species to such a form, then we consider that an algebraic proof of the bijection between $\mathbf{N}$ and the species.

For notational convenience, the following conventions will hold: s, t are atoms (equivalently each unique members of some atomic species), x, y are variables over a semigroup, A, B (capital roman) are species, $A(x), B(x)$ (capital roman functions) are polynomials over x with coefficients from $\mathbf{N}$.

The first species we will consider are those defined by a tree-like structure, by the set operations over atoms of disjoint union, concatenation, and unbounded sequence. We write these operations $A + B, A \cdot B, A^*$.

THEOREM 3.1. *If there is a bijection from both A and B to $\mathbf{N}$, A nd B having respective generating functions $A(x)$ and $B(x)$, then there is a bijection between the species $A + B, A \cdot B, A^*$ and $\mathbf{N}$ using the corresponding generating functions $A(x) + B(x), A(x) \cdot B(x), (1 - A(x))^{-1}$.*

Proof. First we will treat series with infinite support corresponding to infinite sets. Since we are dealing with finite length words over finite alphabet, all coefficients in our polynomials will be positive integers. We'll use the following lemma to help deal with finite series.

LEMMA 3.2. *If a power series $A(x)$ has infinite support (an infinite number of nonzero terms), then there is a bijection between it and $\mathbf{N}$.*

Proof. Suppose A is a set with corresponding generating functions $A(x)$ as encoded with the atom 'x':

$$A(x) = \sum_{k \geq 0} a_k x^k$$

Let A_k be the subset of A whose objects have exactly k instances of the atom x in them, corresponding to the generating function interpretation that a_k is the number of objects of size k. So $A = A_0 + A_1 + \ldots$.

Then the corresponding encoding is

$$\begin{aligned}(A_0 + A_1 + \ldots)(s) &= A_0(s) + s^{a_0} A_1(s) + s^{a_0 + a_1} A_2(s) + \ldots \\ &= \sum_{k \geq 0} s^{a_0 + \cdots + a_{k-1}} A_k(s) \\ &= \frac{s^{(a_0 + \cdots + a_k)} - 1}{s - 1} \\ &= \sum_{0 \leq k < a_0 + \cdots + a_k} x^k,\end{aligned}$$

□

This lemma allows us to deal with individual coefficients of a generating function rather than the whole series when convenient.

Disjoint union of two sets is enumerated by the sum of their generating functions: since $(A + B)(x) = A(x) + B(x) = \sum_{k \geq 0} (a_k + b_k) x^k$, the bijection for objects of size k gives:

$$\begin{aligned}(A_k + B_k)(s) &= A(s) + s^{a_k} B(s) \\ &= \frac{s^{a_k} - 1}{s - 1} + s^{a_k} \frac{s^{b_k} - 1}{s - 1} \\ &= \frac{s^{a_k + b_k} - 1}{s - 1}\end{aligned}$$

The product of two sets has generating function: $(A \cdot B)(x) = \sum_{k\geq 0}\sum_{j\leq k} a_j b_{k-j} x^k$, so the bijection is:

$$\begin{aligned}\left(\sum_{j\leq k} A_j B_{k-j}\right)(s) &= \sum_{j\leq k}(A_j B_{k-j})(s) s^{(a_0 b_k + a_1 b_{k-1} + \cdots + a_j b_{k-j})} \\ &= \sum_{j\leq k}(A_j B_{k-j})(s) s^{a_0 b_k + \cdots + a_k b_0} \\ &= \frac{s^{a_0 b_k + \cdots + a_k b_0} - 1}{s-1}\end{aligned}$$

where $(A_j B_{k-j})(s)$ is the set of $a_j b_{k-j}$ ordered pairs where the first item is from A_j and the second from B_{k-j}.

Since $(A^*)(x) = 1 + A(x) \cdot (A^*)(x) = 1 + \sum_{k\geq 1} x^k \sum_{1\leq j\leq k} a_j a^*_{k-j}$,

$$\begin{aligned}\left(\sum_{j\leq k} A_j A^*_{k-j}\right)(s) &= \sum_{j\leq k}(A_j A^*_{k-j})(s) s^{a_0 a^*_k + \cdots + a_j a^*_{k-j}} \\ &= \sum_{j\leq k}(A_j B_{k-j})(s) s^{a_0 b_k + \cdots + a_j b_{k-j}} \\ &= \frac{s^{a_0 a^*_k + \cdots + a_j a^*_{k-j}} - 1}{s-1}\end{aligned}$$

□

Note that a_k must be 0 for $A^*(x)$ to converge, meaning that if A contained ϵ, A^* would have an infinite number of derivations of ϵ and all other objects. A convergent sequence of operations on polynomials is defined as one for which for all terms of the polynomial there is a species is defined as a finite number of objects of given size

The unranking algorithms are quickly obtained from the bijections above. To construct the object of rank i of size n, one need only compute the coefficient for those species of size n in order, stopping with the greatest value less than i. Then compute the sub-objects using that index and the left over rank. For example, suppose we have two species over x with generating functions:

$$\begin{aligned}A(x) &= <1, 2, 3, 5, \ldots>, \\ B(x) &= <1, 2, 4, 8, \ldots>\end{aligned}$$

Then, whatever construction corresponds to A and B, concatenation of an object from A to an object of B is described by the product of the generating functions:

$$A(x) \cdot B(x) = <1, 4, 11, 27, \ldots>$$

Out of the 27 objects of size $n = 3$, let us find the twentieth object (let $i =$ rank 19 starting from 0) as follows. By convolution, the third coefficient equals $a_0 \cdot b_3 + a_1 \cdot b_2 + a_2 \cdot b_1 + a_3 \cdot b_0$. The partial sums of this series for computation of 27 are

$$0 = 0$$

perm group G_n	generating function: cycle index $Z(G_n)$	$\bigcup_{n\geq 0} G_n$	$\sum_{n\geq 0} Z(G_n)$
E_n	$A(x)^n$	Seq	$\frac{1}{1-A(x)}$
S_n	$\sum_{e\in\lambda(n)} \prod_k \frac{A(x^k)^{e_k}}{k^{e_k} e_k!}$	MSet	$\exp \sum_{n\geq 1} \frac{A(x^n)}{n}$
$A_n - S_n$	$\sum_{e\in\lambda(n)} \prod_k \frac{(-1)^k A(x^k)^{e_k} n!}{k^{e_k} e_k!}$	Set	$\exp \sum_{n\geq 1} \frac{(-1)^{n+1} A(x^n)}{n}$

TABLE 4.1
Generating functions for permutation groups and sets of permutation groups

$$\begin{aligned} 8 &= 1\cdot 8 \\ 16 &= 1\cdot 8 + 2\cdot 4 \\ 22 &= 1\cdot 8 + 2\cdot 4 + 3\cdot 2 \\ 27 &= 1\cdot 8 + 2\cdot 4 + 3\cdot 2 + 5\cdot 1, \end{aligned}$$

the object of rank 19 falling in the third position. The index of that partial sum is $k = 2$, with rank $i' = 19 - 16 = 3$ left over, $3 \cdot 2$ originating from $a_2 \cdot b_1$. This means that the object corresponding to $i = 19$ is a concatenation of some object from A of size 2 and some object from B of size 1. So the object sought is the concatenation of the object from A of size $k = 2$ of rank $\lfloor i'/b_{n-k}\rfloor = \lfloor 3/2\rfloor = 1$ and from B of size $3 - k = 1$ of rank $i' \bmod b_{n-k} = 3 \bmod 2 = 1$.

It is easy to see that addition takes constant time and both multiplication and sequence take linear time to compute, if the underlying generating functions have been computed. The complexity of the integer multiplications is not taken into account.

4. Objects invariant under transformation. With regular expressions, the operations are limited. Common species cannot be expressed with these. Some species can be described using invariance under transformation, that is an object in the species is an equivalence class (or really an example from that class) generated by some transformation. The transformation we consider are actions by some permutation group on tuples of objects.

Zimmerman [15] introduced the concept of operators that enumerate construct classes within which objects are invariant under permutations. These operator also construct canonical examples from these classes, those objects nonisomorphic under the transformation. The basic operations are multiset or bag (a set that allows repeats), and set, with corresponding operators MSet() and Set(). The enumerating generating functions for these come from a straightforward application of Polyá enumeration for the groups S_n the symmetric group (of all permutations), and $A_n - S_n$ the group of one-to-one functions, and then summing over all n (see Harary [6, Chap. 2] for details). Notice that in this framework, a sequence of products of length n, i.e. an n-tuple, corresponds to the operator from the trivial permutation group on n, and that $*()$ is the set of all finite such tuples. We will sometimes call this operator Seq() or Tuple(). A summary of these facts appears in Table 4.

The general method we follow is to determine the coefficients in the expanded generating function, reducing to constructible operations.

For multiset (MSet in Table 4), we compute the coefficients of the inner summation:

$$\begin{aligned} g(x) &= \sum_{k\geq 1} \frac{1}{k} f(x^k) \\ g_k &= \sum_{i|k} \frac{1}{i} f_{k/i}, \end{aligned}$$

and then the coefficients of the exponentiation using logarithmic differentiation:

$$\begin{aligned} h(x) &= \exp g(x) \\ [\log h(x)]' &= [\log \exp g(x)]' \\ h'(x)/h(x) &= g'(x) \\ h'(x) &= h(x)g'(x) \\ \sum_{0\leq k}(k+1)h_{k+1}x^i &= \sum_{0\leq k} h_k x^k \cdot \sum_{0\leq k}(k+1)g_{k+1}x^k \\ &= \sum_{0\leq k}\ \sum_{0\leq j\leq k} (j+1)h_{k-j}g_{j+1}x^k \end{aligned}$$

and solving for h_k:

$$\begin{aligned} h_0 &= 1 \\ h_k &= \frac{1}{k} \sum_{1\leq j\leq k} jh_{k-j}g_j \\ &= \frac{1}{k} \sum_{1\leq j\leq k} jh_{k-j} \sum_{i|j} \frac{1}{i} f_{j/i} \\ &= \frac{1}{k} \sum_{1\leq j\leq k} h_{k-j} \sum_{i|j} if_i \qquad (4.1) \\ &= \sum_{\lambda\in\lambda(k)} \prod_{j\in\lambda} \binom{f_j+e_j-1}{e_j} \qquad (4.2) \end{aligned}$$

where $\lambda(k)$ is the set of integer partitions of k, λ is the multiset of integers representing the partition, $j \in \lambda$ is an integer that appears in λ, and e_j is the number of times that j appears in λ. The factor $\binom{f_j+e_j-1}{e_j}$ corresponds to choosing e_j objects (with repetition, i.e. a multiset) from a set of size f_j, which is what we expect: those multisets of A of size k are exactly those collections of objects from A whose total weight is k, and of those whose weight is $j \leq k$ there may be repetitions. An integer partition has unordered elements that add up to the given weight; the binomial factor then counts the ways for the particular pattern given by that partition, thus establishing the bijection. Note that we do not stop the derivation of coefficients at equation 4.1, because of the division. We seek operations for which bijections with integers can be made, and by itself division is not one of those operations.

This is done as follows. Since we can calculate sums and products, and we can iterate over integer partitions using a constant time 'next' function (Nijenhuis and Wilf

[12]), all that is needed is to find a bijection for the 'choose' operator. To do this we use Pascal's identity $\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r}$, and the identities $\binom{n}{r} = \frac{n}{r}\binom{n-1}{r-1}$ and $\binom{n}{r} = \frac{n}{n-r}\binom{n-1}{r}$. If the index i to be unranked is less than the first summand of Pascal's identity, then an element of A of size k (corresponding to f_k) with index n is added to the set. The rest of the set is then constructed corresponding to the first or the second summand. The construction is $O(r)$ since we can initially compute choose in $O(r)$ using either of the two identities, then an $O(1)$ update of the choose for each element added using the same identities.

For Set, the generating function has coefficients only slightly different from MSet:

$$\begin{aligned} h_0 &= 1 \\ h_k &= \frac{1}{k} \sum_{1 \le j \le k} h_{k-j} \sum_{i|j} (-1)^{(j/i)+1} i f_i \\ h_k &= \sum_{\lambda \in \lambda(k)} \prod_{j \in \lambda} \binom{f_j}{e_j} \end{aligned}$$

which chooses sets of size e_j out of f_j without repetition. The proof, bijection, and implementation are similar to that for multisets.

Notice that for sets and multisets, the particular bijection is dependent on some ordering of integer partitions of k. Since there are many of these, we choose the common reverse lexicographic ordering. This will affect the average number of steps in construction of objects from a recursively defined species.

Often, a bag or set with exactly d elements is sought. For a specific length, the calculations are only slightly different from the unbounded versions of MSet and Set. For a bags or sets of exactly length d [1], one uses multisets and combinations again, but only those arising from integer partitions having d or fewer parts. Since d is a constant, the base generating function can have objects of zero weight and these will fill out the rest of the tuple when the partition has strictly fewer than d parts. For tuples of length d with no invariants, the easy correspondence with enumeration does not hold because the calculation uses division (see Wilf [12, chap. 21]). Repeated squaring gives an $O(n^2 \log n)$ preprocessing time for construction though. Table 4 summarize the above computations; compare with Table 4.

As to implementation of these operations, many optimizations can be made: 1) If it is known that one want to generate many items no greater than n, all enumeration calculations can be done ahead of time (preprocessing). 2) Construction can be sped up considerably (average case) if partial sums are computed; this will allow binary search. 3) Since coefficients can be quite large, efficient infinite precision integer routines would be helpful (e.g. Schönhage-Strassen integer multiplication). 4) Straightforward FFT $O(n \log n)$ polynomial multiplication could be used but only as long as sufficient floating point precision is available. If all coefficients are guaranteed to be less than a certain value, then number theory FFT using primitive roots would be better.

[1] Tuples of length exactly d invariant upon permutation, with and without repetition respectively; d refers to the degree of the group, the size of the set being acted upon.

operation	constructible coefficient of x_n
A(x)+B(x)	$a_n + b_n$
$A(x) \cdot B(x)$	$\sum a_k b_{n-k}$
$A(x)^d$	repeated squaring
d even	$(A(x)^{d/2})^2$
d odd	$A(x) \cdot A(x)^{d-1}$
Seq$(A(x))$	$\sum_{1 \le k \le n} a_k a^*_{n-k}$
MSet$(A(x))$	$\sum_{\lambda} \prod \binom{a_k + e_k - 1}{e_k}$
Set$(A(x))$	$\sum_{\lambda} \prod \binom{a_k}{e_k}$

TABLE 4.2
Coefficients and constructors for operators

5. Solution of system of equations. The specification of a species is made by a set of implicitly defined equations which may be mutually referring. The above examples and similar species are traditionally solved through ad hoc methods of manipulating generating functions. Common methods are Lagrange inversion, calculating roots, or analogous methods from solving differential equations.

If we restrict the constructor operations to union and concatenation of atoms to variables, then one can find a closed form description of the species in terms of union, concatenation and sequence. For example, the following system :

$$\begin{aligned} X &= \epsilon \\ X &= aY \\ X &= bY \\ Y &= a \\ Y &= bX \\ Y &= aY \end{aligned}$$

(where a and b are atomic species) defines the two species A and B. This is easily converted to the right-linear system:

$$\begin{array}{ccccccccc} X & = & \epsilon & + & & & + & (a+b) & Y \\ Y & = & a & + & b & X & + & a & Y \end{array}$$

which, with matrix-vector product defined as $(S \cdot T)_{ij} = \sum_k S_{ik} \cdot T_{kj}$, by the algebraic properties of the abstract operations $+$ and $\cdot$, can be written as:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \epsilon \\ a \end{bmatrix} + \begin{bmatrix} \emptyset & a+b \\ b & a \end{bmatrix} \cdot \begin{bmatrix} X \\ Y \end{bmatrix}$$

Use the operations of union, concatenation by an atom on the left, and recursion, this is a linear system of equations which is be solved completely as:

$$x = a + Ax \to x = aA^*$$

where A^* is the Kleene closure of the matrix computed by the Floyd-Warshall-Kleene algorithm which operates over closed semirings [10]. The solution to the above is then:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} \epsilon + (a+b)^2(ba+bb+a)^* \\ (a+b)(ba+bb+a)^* \end{bmatrix}.$$

This means that, by eliminating all variables, we can convert an implicit description (the system of equations) into a regular expression using only atoms, union, concatenation and Kleene closure. Since Kleene closure is itself definable as a right-linear equation, namely $A^* = \epsilon + A \cdot A^*$, (which is right-linear if A is right-linear), then this operation can also be used in a description and be solvable. Likewise, any operation, for which regular languages are closed, (e.g. complement, intersection, all prefixes, etc.), can be used in a description and the system will remain solvable.

6. Multivariate systems. We have put off accounting for multivariate polynomials, that is species with more than one atom (letter in the alphabet), because proofs would be intolerably cumbersome. However, it is easy to see that they would still hold for a finite set of variables by an inductive method.

THEOREM 6.1. *There is a bijection between the natural numbers and species over a finite number of atoms.*

Proof. The base cases on a single variable have been done above. The inductive case constructs a bijection similar to that for multiplication above (likewise similar to the natural bijection between $\mathbf{N}$ and $\mathbf{N} \times \mathbf{N}$). □

If all we care for is counting and constructing objects with total weight n, we need not maintain information for every atom, we only need use a one-dimensional array. However, for maintaining multiplicity for each different atom, we need to maintain a vector of multiplicity for each possible term in the generating function. For total weight n and k atoms, the number of terms is the number of distinct tuples each of whose sum of entries is $\leq n$, which is the number of integer compositions $\leq n$ with k parts: $\sum_{0 \leq j \leq n} \binom{j+k-1}{j} = \binom{n+k}{n}$ which is $O(n^k)$ for constant k and $O(4^n/\sqrt{\pi n})$. for $k = O(n)$, all impractically large.

7. Context–free species. We still need to address context free species, those implicitly describable by a set of mutually referring definitions. The restriction on the set is that each generating function must appear at least once as a monic term of degree one. This corresponds to the description of a context free grammar as a grammar with the left hand side of a production being a single variable.

It is important that the grammars here are not necessarily unambiguous. Often a grammar can be used to count and construct a species with different derivations of the same string of terminals, for example, the Catalan correspondence to the ways of parenthesizing a word of length n. If the species sought must have unique derivations for objects corresponding to the string consisting of the leaves of an ordered tree, then its grammar must be specifically designed to be unambiguous. This distinction between words and derivations comes from the distinction between languages and numbers: since the union of sets is idempotent ($a = a \cup a$) but addition in $\mathbf{N}$ is not ($a \neq a + a$), the polynomial calculations for counting will not collapse different parsings of the same string.

A context free grammar, which uses concatenation, alteration, and recursion, implicitly defines a species, the atoms of which are the terminal symbols of the grammar. The grammar corresponds then to a system of nonlinear equations: the terminal symbols of the grammar correspond to constant coefficients in the system, the variable symbols of the grammar correspond to polynomials over the terminals, and the language recognized by the grammar corresponds to the solution of the system (or least fixed point) in terms of these polynomials. Gröbner basis calculation (via Buchberger's algorithm, see Becker and Weispfenning [2]) can eliminate as many variables as possible. This calculation can reduce the set of polynomials (corresponding to the grammar) but it is not guaranteed to solve the system and itself is quite a costly operation (doubly exponential). Though all but one variable be eliminated, the remaining polynomial may be of too high a degree to solve tractably.

This lack of solvability in general means we must rely on finite approximations. The grammar induces a graph of dependence, possibly with cycles. If cycles exist, then only partial solutions (in terms of coefficients of the resulting power series) can be calculated. This is performed in a traditional manner for any finite approximation: set all variables to zero, then iterate the computation. For an approximation to words of size n, the iteration must be n times to guarantee that the coefficients up to degree n are correct. This assumes that the system is convergent, which is ensured by following the prescriptions of the operations, e.g. Set cannot be applied to a species with ϵ.

This situation poses several problems. We would like to find a formal language characterization of the set operations when added to context free operations. On the other hand, we would like to find a combinatorial interpretation for context sensitive construction rules, and if there is a set of group invariant set of operations that, when used with context free rules, yields the set of context sensitive species. Also, we conjecture that a general algorithm exists to construct any species that is invariant over a permutation group (specified by generators).

8. Examples and Constructions. Here are some examples of species with context–free presentations:

species	description	grammar		
C	subsets of a set of size n	$S \leftarrow \overbrace{AA\cdots A}^{n} = A^n,$ $A \leftarrow \epsilon \mid x$	$=$	$(1+x)^n$
F	sequences of 1's and 2's	$S \leftarrow \epsilon \mid xS \mid xxS$	$=$	$(x+x^2)^*$
B	binary trees	$S \leftarrow \epsilon \mid xSS$		

and here are their corresponding enumerating formulas:

species	recurrence	generating function	generating function recurrence
C	$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$	$(1+x)^n$	$C_n(x) = (1+x)C_{n-1}(x)$ $C_0(x) = 1$
F	$F_n = F_{n-1} + F_{n-2},$ $F_0 = F_1 = 1$	$\frac{1}{1-x-x^2}$	$F(x) = 1 + xF(x) + x^2F(x)$
B	$B_n = \sum_{k=0}^{n-1} B_k B_{n-k-1}$	$\frac{1-\sqrt{1-4x}}{2}$	$B(x) = 1 + xB(x)^2$

Rooted trees of n nodes with unique unordered children (rooted unordered identity

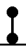
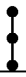
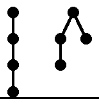
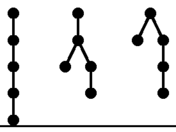
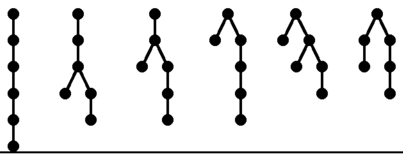
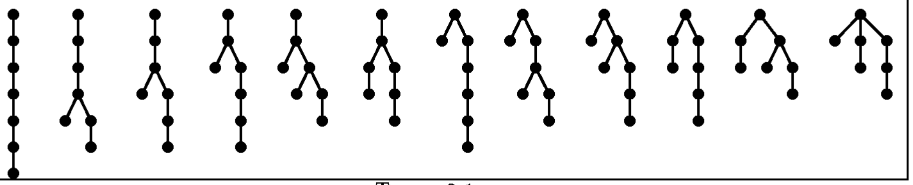
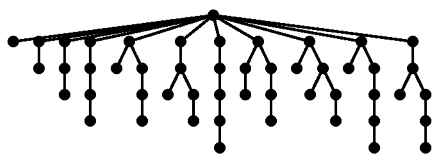
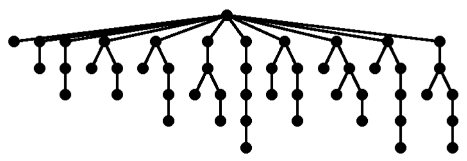
1	
2	
3	
4	
5	
6	
7	

TABLE 8.1
Rooted trees up to size 7



114045563401548309



114045563401548310

TABLE 8.2
The last two rooted trees of size 50

trees) are described syntactically as an atom (the root) of weight 1 followed by a set of children which are also such objects. Its construction is described by $R = x \cdot \mathrm{Set}(R)$, which immediately gives the implicit generating function $R(x) = x \cdot \exp \sum_{k \geq 0} R(x^k)/k$ whose coefficients for x_0 through x_7 and x_{50} are calculated by the above procedures to be $\langle 0, 1, 1, 1, 2, 3, 6, 12, \ldots, 114, 045, 563, 401, 548, 311, \ldots \rangle$. This is sequence A004111/M0796 in the Encyclopedia of Integer Sequences [14]. The twelve rooted unordered identity trees up to size 7 are shown in Table 8.1. The last two trees of size 50 with the given ordering are shown in Table 8.2. Code to compute and construct such examples is available at `http://www.uiuc.edu/ph/www/maharri/co.tar.gz`.

9. Conclusions. An efficient bijection between the natural numbers and a combinatorial species also gives solutions to other combinatorial algorithms. Nijenhuis and Wilf [12] describe four general operations on species: ranking, unranking, random selection and next in sequence. The method above is an unranking procedure; given a natural number, we can create a particular object of a species. Ranking returns the number for the object, this is the inverse operation or unranking. An object can be selected uniformly at random using the unrank procedure by selecting an integer uniformly at random within any subset (the most likely subset being those objects of the same size). The next object in sequence after some object c in a total ordering of its species is given by Unrank(Rank(c) + 1). The pattern of this sequence is highly dependent on the implementation of polynomial operations, i.e. there is no guarantee of hamiltonicity (associated with Gray code sequencing). Also, because of the generality of the procedure, this method is unlikely to be the most efficient for constructing many species, in that there may be sequencing operations that have better amortized complexity (see Savage's survey [13]).

REFERENCES

[1] L. Alonso and R. Schott. *Random generation of trees.* Kluwer Academic Publishers, Boston, MA, 1995. Random generators in computer science.

[2] T. Becker and V. Weispfenning. *Gröbner bases*, volume 141 of *Graduate Texts in Mathematics.* Springer-Verlag, New York, 1993. A computational approach to commutative algebra, In cooperation with Heinz Kredel.

[3] F. Bergeron, G. Labelle, and P. Leroux. *Combinatorial Species and Tree-like Structures*, volume 67 of *Encyclopedia of Mathematics and its Applications.* Cambridge University Press, Cambridge, UK, 1998.

[4] N. Chomsky and M. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Languages*, pages 118–161. North-Holland, Amsterdam, 1963.

[5] P. Flajolet, P. Zimmermann, and B. V. Cutsem. A calculus for the random generation of labelled combinatorial structures. *Theoretical Computer Science*, 132(1):1–35, Sep 1994.

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

[7] T. Hickey and J. Cohen. Uniform random generation of strings in a context-free language. *SIAM J. Comput.*, 12(4):645–655, 1983.

[8] M. R. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theoret. Comput. Sci.*, 43(2-3):169–188, 1986.

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

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

[11] H. G. Mairson. Generating words in a context-free language uniformly at random. *Inform. Process. Lett.*, 49(2):95–99, 1994.

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

[13] C. Savage. A survey of combinatorial gray codes. *SIAM Review*, 39(4):605–629, Dec. 1997.

[14] N. J. A. Sloane and S. Plouffe. *The Encyclopedia of Integer Sequences.* Academic Press Inc., San Diego, CA, 1995.

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