Algorithms Seminar 1999–2000,
F. Chyzak (ed.), INRIA, (2000), pp. 71–74.

Available online at the URL
http://algo.inria.fr/seminars/.

Attribute Grammars and Automatic Complexity Analysis

Marni Mishna
Algorithms Project, INRIA Rocquencourt

June 19, 2000

Summary by Marianne Durand

Abstract

Starting from combinatorial structures, one can study some of their characteristics by means of attribute grammars [1, 2]. This leads to multivariate generating functions that permit us to study the distribution of these characteristics, part of it automatically.

1. Attribute Grammars

The grammars considered here are built from atoms, Z, Z_1,... of weight 1 and from an ϵ of weight 0. The production rules are described in terms of a few constructors: union, cartesian product, set, sequence and cycle. These constructors can take place in a labelled world (permutations) or unlabelled (trees) and they are already present in the COMBSTRUCT package. A grammar is composed of production rules of the type $T = \Phi(T_1, \ldots, T_n)$; T is said to be an ancestor of each T_i and each T_i is a descendant of T. The attributes on these grammars are values on the objects produced by the grammar, here on combinatorial structures, like for example the size or the internal path length on a binary search tree. An attribute is *synthesized* if it is a function of his descendants (size of a tree) and *inherited* if it is a function of his ancestors. An example of an inherited attribute is the depth of a tree. The depth is defined by : the depth of the root is zero and the depth of a subtree is the depth of its father plus one. An attribute is *well-defined* if there are no circular dependencies amongst the attributes, which can be checked algorithmically [5]. The attribute is *linear* if it is a linear function of the attributes of the descendants. The size of a tree is a linear attribute, but the height of a tree defined by the maximum of the height of the subtrees plus one is not.

We now consider linear synthesized and well-defined attributes. The general specification of a structure is:

$$B = \Phi_1(B_1^1, \ldots, B_{k_1}^1) \mid \ldots \mid \Phi_n(B_1^n, \ldots, B_{k_n}^n). \tag{1}$$

where Φ_i is a standard constructor, like cartesian product, set, sequence, or cycle, or a terminal. The general form of the definition of an attribute F_i then is

$$F_i(B) = \bigcup_{1 \le m \le n} \phi\left(\delta_i^m + \sum_{j,k} \alpha_{i,j}^m F_j\left(B_k^m\right)\right) + \gamma_i$$

where lower case indexed greek letters indicate integer constants, and F_j corresponds to other attributes. The letter ϕ stands for a general iterative operator coding the fact that all the subelements of the structure are considered. For example if Φ is the sequence constructor, each element of the sequence is considered recursively. The non-planar trees are defined by $T = N \cdot \mathrm{Set(T)}$ and there

the internal path length is specified by $\mathrm{ipl}(T) = \phi(\mathrm{size}(T) + \mathrm{ipl}(T))$. Other examples are the area below Dyck paths, the number of cycles in a permutation or the number of parts in a partition.

All these attributes can be encoded in multivariate generating functions as follows. If the attributes are named F_i and the structure is defined as in equation (1), the generating function in an unlabelled world is

$$B(z_0, \ldots, z_k) = \sum_{b \in B} z_0^{|b|} z_1^{F_1(b)} \ldots z_k^{F_k(b)}. \tag{2}$$

Let $\mathbf{z}$ be the vector $(z_1, \ldots, z_k)$, α^m be the matrix $[\alpha_{i,j}^m]$, γ^m and δ^m be vectors, where m is an index indicating the related constructor Φ_m. We use the following notations: $\mathbf{z}^\delta = (z_1^{\delta_1}, \ldots, z_k^{\delta_k})$ and $\mathbf{z}^\alpha = \left(z_1^{\alpha_{1,1}} \ldots z_k^{\alpha_{1,k}}, \ldots, z_1^{\alpha_{1,k}} \ldots z_k^{\alpha_{k,k}}\right)$. This allows us to state the Attribute Grammars Generating Function theorem.

Theorem 1. [6] *Given the grammar specification* $B = \Phi_1(B_1^1, \ldots, B_{k_1}^1) \mid \ldots \mid \Phi_n(B_1^n, \ldots, B_{k_n}^n)$ *where each* Φ_i *is a grammar constructor or a terminal and given the set of attribute productions* $F_i(B) = \bigcup_{1 \le m \le n} \phi\left(\delta_i^m + \sum_{j,k} \alpha_{i,j}^m F_j\left(B_k^m\right)\right) + \gamma_i$ *the multivariate generating function* $B(\mathbf{z})$ *satisfies*

$$B(\mathbf{z}) = \sum_m \mathbf{z}^{\gamma^m} \mathcal{G}_{\Phi_m}\left(\mathbf{z}^{\delta^m} B_k^m(\mathbf{z}^{\alpha^m})\right)$$

where $\mathcal{G}_{\Phi_m}$ *is the classical generating function transformation on structures.*

Proof. The proof requires a study of each constructor. We give here a simplified proof where $B = \Phi(C)$. As in equation (2) the generating function is defined by

$$B(\mathbf{z}) = \sum_{b \in B} z_1^{F_1(b)} \ldots z_k^{F_k(b)}.$$

By replacing with the definition of F_i, i.e., $F_i(B) = \phi\left(\delta_i + \sum_{j,k} \alpha_{i,j} F_j(B_k)\right) + \gamma_i$, we obtain

$$B(\mathbf{z}) = \sum_{b \in B} \prod_l z_l^{\gamma_l} \cdot \prod_{a \in b} \prod_i z_i^{\delta_i + \sum_{j=1}^k \alpha_{ij} F_j(b)}, \tag{3}$$

which simplifies into

$$B(\mathbf{z}) = \mathbf{z}^\gamma \sum_{b \in B} \mathbf{z}^\delta \prod_{a \in b} \prod_j \left(\prod_i z_i^{\alpha_{ij}}\right)^{F_j(b)}.$$

In view of $C(\mathbf{z}) = \sum_{c \in C} \prod_j z_j^{F_j(c)}$ and $B(z) = \sum_{b \in B} \prod_{a \in b} z^{|b|} = \mathcal{G}\left(B(z)\right)$, we now have the final result

$$B(\mathbf{z}) = \mathbf{z}^\gamma \mathcal{G}_\Phi\left(\mathbf{z}^\delta C(\mathbf{z}^\alpha)\right).$$

We obtain a simple formula to express the generating function of a structure given the type of its attributes.

□

2. Automatic Complexity Analysis

The idea of working on combinatorial properties is not new, it has already been exploited in $\Lambda\Upsilon\Omega$ [3, 7], part of which is implemented in the COMBSTRUCT package. Given a combinatorial structure and a class of algorithms based on programming primitives like sequence of programs, test on unions, partial program descent and full component iteration, $\Lambda\Upsilon\Omega$ returns the asymptotic value of the cost of the program on all structures of size n. It is then possible to get the average value of the cost of the considered program. The programs analysed by $\Lambda\Upsilon\Omega$ can be viewed as attributes

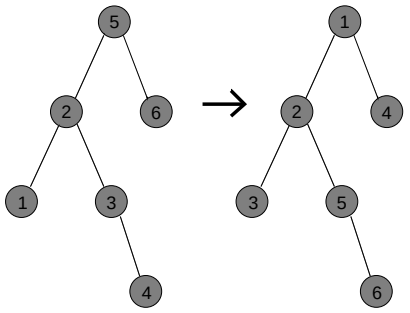

FIGURE 1. The binary search tree and increasing tree associated with [521634].

on a grammar corresponding to the structure. In fact the expressivity of $\Lambda\Upsilon^{\Omega}$ is encompassed by the attribute grammar system. The attribute grammars are well implemented and will be in the COMBSTRUCT package soon. For example it is possible to compute the cost of differentiating a regular expression based on plus, times and exp and to get the average and the variance of this cost, which is not possible in $\Lambda\Upsilon^{\Omega}$.

These techniques can also be applied to other constructors, if their translation into generating function is known. For instance the Quicksort algorithm can be studied using attribute grammars. The Quicksort algorithm takes as input a random permutation, chooses a pivot, sorts the elements according to their position with respect to the pivot and then sorts recursively the two subarrays. The run of the algorithm can be visualised by a binary search tree, the root being the pivot, and the two sons being the two subarrays. The complexity is the number of comparisons done, which corresponds to the internal path length of the binary search tree. This correspondance between executions of the algorithm and binary search trees is not a bijection, because the inputs 231 and 213 yield the same tree. The solution to this problem is to keep the shape of the tree and to label it with the order in which the nodes are filled, as shown on Figure 1. This gives a bijection between runs of Quicksort and increasing trees. To describe increasing trees with attribute grammars, we need to introduce the *Greene operator* also called *box operator* [4]. In a labelled structure, the Greene operator specifies where the minimum label is to be. For example the increasing trees are defined by $T = \epsilon \mid T_1 \cdot \mathrm{Min}(N) \cdot T_2$ which specifies that the minimum is in the root N. The generating function has been determined by Greene:

$$T(z) = \int_0^z T^2(x) \frac{\partial N(x)}{\partial x}\, dx.$$

It is now possible to define the internal path length as an attribute on the increasing tree structure by the relation

$$\mathrm{ipl}(T) = 0 \mid \mathrm{ipl}(T_1) + \mathrm{size}(T_1) + \mathrm{ipl}(T_2) + \mathrm{size}(T_2),$$

assuming that the internal path length of a node is 0, which is coherent with the complexity model of the number of comparisons. The multivariate generating function is

$$T(z,u) = 1 + \int_0^z \left(\frac{\partial}{\partial x} x\right) T(xu,u)^2\, dx.$$

The average is therefore

$$\frac{[z^n]T_u(z,u)|_{u=1}}{[z^n]T(z,1)} = 2H_n - 3 + \frac{H_n}{n} \quad \text{with} \quad T(z,1) = \frac{1}{1-z}$$

where H_n is the nth harmonic number.

All this work has been implemented in Maple in such a way that the syntax of attributes grammars use the same basic functions as COMBSTRUCT. For example if a grammar rule is $A = B \mid C$ then an attribute for A follows the equation, in COMBSTRUCT syntax,

```
F(A)=Union(b_1*F_1(B)+...+b_k*F_k(B),c_1*F_1(C)+...+c_k*F_k(C))
        + a_1*F_1(A)+...+a_k*F_k(A)+a_0.
```

Similar rules apply for product and set. Since COMBSTRUCT can verify if a grammar is well defined, the same algorithm can tell if an attribute grammar is linear and synthetic. For example if one looks again at the internal path length but this time of a binary Catalan tree, using two lines to define the grammar ($B = \epsilon + zB^2$) and the attribute coding internal path length ($\mathrm{ipl} = \mathrm{size}(B) + \mathrm{ipl}(B_1) + \mathrm{ipl}(B_2)$) and five to compute the generating functions and the first moments, one gets automatically that the average equals $\sqrt{\pi}n^{3/2}+O(n)$ and the variance equals $(10/3-\pi)n^3+O(n^{5/2})$. This computation can also be done on examples like the grammar defining the expressions based on zero, one, x, sum, product and exponentiation. It is possible to define the attribute coding the size of an expression after differentiation. This leads to an automatic proof that on average differentiating an expression of size n yields an expression of size $0.8\,n^{3/2}$.

Attribute grammars provide a good way of describing recursive properties of decomposable structures; a structure is decomposable if it can be expressed with basic atoms (ϵ, Z) and basic constructors (union, product, set, sequence, ...). The work that has been done on this subject can be used to obtain algorithms for random generation of structures with given attribute value, and also to obtain the distribution of the attribute. It can be continued on other attribute types for example heads or tails of sequences. From the aspect of attribute grammar research, some theory has been developed on the idea of coupling grammars. This simulates repeated application of a function. This, for example, would allow a simple analysis of repeated differentiation, and other composed functions. This requires a system where the attributes may be more than constants, but rather further structures.

Bibliography

[1] Delest (M.-P.) and Fedou (J. M.). – Attribute grammars are useful for combinatorics. *Theoretical Computer Science*, vol. 98, n° 1, 1992, pp. 65–76. – Second Workshop on Algebraic and Computer-theoretic Aspects of Formal Power Series (Paris, 1990).

[2] Dutour (I.) and Fédou (J. M.). – Object grammars and random generation. *Discrete Mathematics and Theoretical Computer Science*, vol. 2, 1998, pp. 47–61.

[3] Flajolet (P.), Salvy (B.), and Zimmermann (P.). – *Lambda-Upsilon-Omega: the 1989 cookbook.* – Research Report n° 1073, Institut National de Recherche en Informatique et en Automatique, August 1989. 116 pages.

[4] Green (D.). – *Formal languages and their uses.* – Ph. D. Thesis, Stanford University, 1985.

[5] Knuth (D. E.). – Semantics of context-free languages. *Mathematical Systems Theory*, vol. 2, n° 2, 1968, pp. 127–145.

[6] Mishna (Marni). – *Attribute grammars and automatic complexity analysis.* – Research Report n° 4021, Institut National de Recherche en Informatique et en Automatique, October 2000. 20 pages.

[7] Zimmermann (P.). – *Series génératrices et analyse automatique d'algorithmes.* – Ph. D. Thesis, École polytechnique, Palaiseau, 1991.