A complexity calculus for recursive tree algorithms

Philippe Flajolet, Jean-Marc Steyaert

INRIA

CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(3) 954 90 20

Rapports de Recherche

N° 239

A COMPLEXITY CALCULUS FOR RECURSIVE TREE ALGORITHMS

Philippe FLAJOLET
Jean-Marc STEYAERT

Octobre 1983

A COMPLEXITY CALCULUS FOR RECURSIVE TREE ALGORITHMS

Philippe FLAJOLET
I.N.R.I.A.
78150 - Rocquencourt (France)

Jean-Marc STEYAERT
Centre de Mathématiques Appliquées
Ecole Polytechnique
91128 Palaiseau (France)

ABSTRACT :

We study a restricted programming language over tree structures. For this language, we give systematic translation rules which map programs into complexity descriptors. The descriptors are in the form of generating functions of average costs. Such a direct approach avoids the recourse to recurrences ; it therefore simplifies the task of analyzing algorithms in the class considered and permits analysis of structurally complex programs. It also allows for a clear discussion of analytic properties of complexity desciptors whose singularities are related to the asymptotic behavior of average costs. Algorithms that are analyzed in this way include : formal differentiation, tree matching, tree compatibility and simplification of expressions in a diversity of contexts. Some general results relating (average case) complexity properties to structural properties of programs in the class can also be derived in this framework.

RESUME :

L'objet de cet article est l'étude d'un langage de programmation restreint pour les structures arborescentes. On donne pour ce langage des règles de traduction systématiques qui associent aux programmes des descripteurs de complexité représentant les coûts moyens sous forme de série génératrice. Une telle approche directe évite le recours aux récurrences et permet ainsi l'analyse de programmes structurellement complexes. Ceci nécessite l'examen des propriétés analytiques des descripteurs de complexité dont les singularités déterminent le comportement asymptotique du coût moyen. Divers algorithmes sont étudiés de cette manière : dérivation formelle, recherche de motifs, compatibilité, simplification d'expressions. Quelques résultats généraux reliant les coûts moyens à la structure des programmes de la classe définie sont également donnés dans ce cadre.

1. INTRODUCTION

A large number of algorithms of both practical and theoretical interest considered in the literature are realized by combinations of recursive descent procedures over recursively defined data structures. As is also the case for loop-programs, such algorithms have the interesting property that termination is a priori guaranteed by their structure, so that no undecidability question arises to limit the possibility of analyzing program behavior. We mention the following classes :

(i) Tree algorithms : formal differentiation (in formal manipulation systems), tree matching (as occurs in compiler optimization), reduction algorithms (in theorem proving or symbolic execution).

(ii) Comparison searching and sorting : binary search (for maintaining dictionaries), heap-like sorting, tree sorting, quicksort and a number of their variants.

(iii) Digital search : with algorithms for inserting, deleting and querying tables that maintain digital keys (strings over some fixed alphabet) ; various set-theoretic operations such as union and intersection can also be performed efficiently.

The performances of a number of such algorithms have already been analyzed and (ii), (iii) represent a nonnegligible fraction of [Kn73]. A few analyses pertaining to (i) are also discussed in [Kn68]. Existing analyses essentially obey the following paradigm.

For an algorithm A over a set of inputs I (viz. trees, permutations, digital sets...), with I_n the subset of inputs of size n, we consider the quantities :

$$i_n = \text{card} I_n \qquad \tau a_n = \sum_{e \in I_n} \text{time}^A [e]$$

representing the number of possible inputs of size n and the cumulated computation time of A over such inputs. An interesting quantity is the average performance of A (over I_n) defined by

$$\bar{\tau a}_n = \frac{\tau a_n}{i_n} \ .$$

To determine these quantities, i.e., "analyze" the algorithm, one usually sets up recurrence relations based on the one hand on a combinatorial decomposition of the structure into smaller components :

$$i_n = \phi\,(\{i_j\}_{j<n})$$

and, on the other hand, on tracing back the complexity of the algorithms on substructures and subroutines. For instance, in the case of two mutually dependent subroutines, A, B equations would have the form :

$$\tau a_n = \psi_A(\{\tau a_j\}_{j<n} ; \{\tau b_j\}_{j<n} ; \{i_j\}_{j<n})$$

$$\tau b_n = \psi_B(\{\tau a_j\}_{j<n} ; \{\tau b_j\}_{j<n} ; \{i_j\}_{j<n})$$

One then attempts to solve these recurrence relations, relying on classical techniques from the calculus of finite differences, the algebra of formal power series or analysis.

Such performance analyses are basically one-shot. A new set of equations has to be set up for each new algorithm considered and often ad hoc solution methods have to be found in each particular case. Experience, however, shows that most "reasonable" algorithms ultimately exhibit relatively simple behaviors (expressible in terms of standard functions), and the set of applicable techniques seems much more restricted than appears at first glance.

This paper proposes to develop an alternative framework to the analysis of such algorithms, trying to capture many of the regularities encountered in the recurrence approach. It starts with the observation that generating functions for input counts (i_n) and cumulated complexity counts (τa_n) defined by

$$i(z) = \sum_n i_n z^n \qquad \text{and} \quad \tau a(z) = \sum_n \tau a_n z^n$$

satisfy equations whose shapes reflect the structural definitions of inputs and programs (such a relation also exists between programs and recurrences but is of a much less simple form). This observation can be developed into a system of translation rules that allow a systematic translation from program texts into complexity descriptors (the generating functions of the cumulated complexities τa_n). More precisely, each translation rule specifies in a particular context how the complexity descriptor of a larger program can be determined from the complexity descriptors of its simpler components.

The system of rules thus forms the algebraic part of a complexity calculus for the class of programs considered. The next stage, which involves complex analysis, is to recover proper information on program complexity form these complexity descriptors. We use here the existence of relations between the nature of a function around its singularities in the complex plane and the asymptotic behavior of its Taylor coefficients. By tracing singular parts of complexity descriptors, we are thus able to draw, in a systematic way, precise conclusions on the costs of algorithms.

We propose here to illustrate this approach by studying a restricted programming language over tree structures. This language PL-tree operates on tree structures as appear in compiling, formal manipulation systems, theorem proving, automatic inference, i.e., essentially term trees (in single-typed or heteregeneous algebras). It allows for basic recursive descent mechanisms possibly guided by the informations contained in nodes of the trees. The language is powerful enough to include programs for formal differentiation, tree maching, tree enbedding, reduction of expressions for which precise performance estimates are given.

Amongst the works that bear relations to our approach we mention :

(i) investigation of relations between structural properties of combinatorial objects and corresponding generating functions is an important trend in combinatorial analysis : one can refer to works by Rota, Foata, Schützenberger [Ro75][FS70] and the very systematic treatment in Jackson and Goulden [JG81].

(ii) Wegbreit [W76] and Ramshaw [R79] have proposed complexity assertions systems that are analogues of the Floyd-Hoare assertions in formal semantics ; these systems can be used to express computational properties of programs but seem too general to allow automatic performance analysis of specific classes. Closer to our objective is Wegbreit's system [W75] for automating the analysis of a set of LISP-like procedures on lists. It is, however, based on recurrence relations and, as such, has to rely in its resolution part on stringent Markovian approximations to probabilities of test satisfaction leading only to approximate analyses. Several convergent ideas also appear in the work of Burge [Bu75] where various correspondences are indicated between recursive data structures and generating functions.

(iii) We finally exploit informations on functions around their singularities to derive the asymptotics of their coefficients. This is related to the Darboux-Polya method [He74], especially to developments in a paper by Meir and Moon [MM78] , as well as the use of particular contour integration by Odlyzko [O82] and [FO82][SF82].

The plan of the paper is as follows :

Section 2 starts with a description of the programming constructs of PL.tree (Section 2.1.), together with associated complexity rules (Section 2.2.) ; proof techniques for these algebraic complexity rules are discussed in Section 2.3., and the set of analytic tools needed to interpret complexity descriptors appears in Section 2.4.

We then work out several analyses in detail :

Firstly the classical algorithm for symbolic differentiation (Section 3); secondly a tree compatibility algorithm (Section 4) closely related to the "generalization problem" in symbolic manipulation. In Section 5, we conclude with two further examples : tree matching revisited with results complementing thoseof [SF82] and a simple case of a top down recursive simplifier.

2 - COMPLEXITY DESCRIPTORS AND COMPLEXITY RULES

In this section, we first describe informally the main features of the programming language (PL-tree) to which our rules apply. We then state the main rules that allow calculation of complexity descriptors associated to programs ; these rules also serve indirectly as a specification of the allowable constructs of PL-tree. Lastly, we present some analytic translation lemmas that make it possible to extract information on the asymptotic time complexity of programs from the equations satisfied by their complexity descriptors as derived through the complexity rules.

2.1. - A Programming Language on Trees

The basic data type underlying PL-tree is a set of term trees or expression trees : a fixed alphabet of operator symbols Ω is given ; to each operator $\omega \in \Omega$ is associated its arity (or degree) - $\delta(\omega)$- ; this defines in a standard way the set of term trees constructed on Ω : a tree T with root ω can be written in functional form as

$$T = \omega(T_1, \ldots, T_{\delta(\omega)})$$

where the T_i's are themselves trees built on Ω . The set T of all trees built on Ω thus satisfies formally the equation :

$$T = \sum_{\omega \in \Omega} \omega\,(T, T, \ldots, T) \qquad (1)$$

in which the number of T's appearing in $\omega(T,\ldots,T)$ is equal to the arity of ω. Equation (1) will also be written for short as

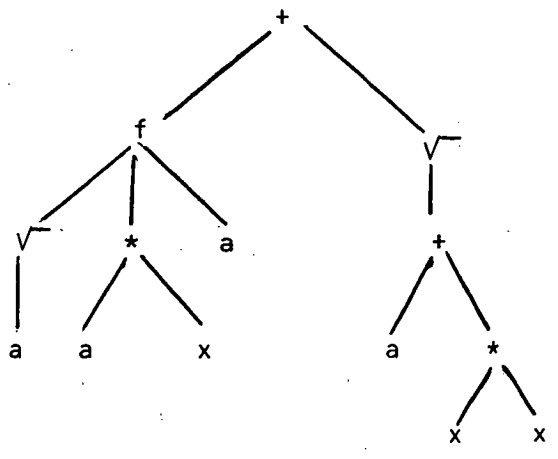
$$T = \Omega\,(T)$$

Our analyses are relative to programs operating on data types T_1, T_2,... (including T itself) which are subtypes of T and are defined by some $\Omega_i \subset \Omega$:

$$T_i = \Omega_i\,(T_i) \qquad (2)$$

Elements of T are commonly written in functional form as above and can be represented as trees as shown in Figure 1.

We shall denote the size of a tree T - i.e. its total number of nodes or equivalently of symbols - by $|T|$.



$$T = +(f(\sqrt{}(a), *(a,x), a), \sqrt{}(+(a, *(x,x))))$$

Figure 1 : A term T and its associated tree corresponding to the symbol alphabet $\Omega = \{a^{(0)}, x^{(0)}, \sqrt{}^{(1)}, +^{(2)}, *^{(2)}, f^{(3)}\}$ (superscripts mark arities).

PL-tree is a language of a procedural type which does not allow explicit assignment of variables. From developments that follow it will be clear that it is not universal. Such a constraint has to be put on the language since, as is well known, complexity properties of universal programming languages are highly undecidable. Yet PL-tree allows programming of a large number of recursive procedures of general interest as it has been mentionned previously.

Without going into a formal specification of the language, that would take us too far away, we briefly discuss its main characteristics :

(i) Basic data types :

The basic type is the set of trees defined in (1) together with subtypes of the form (2). Integers are allowed in a restricted form in the control of loops and in tests on node degrees, but cannot be assigned freely to variables or procedure results.

Booleans may only appear as the result of elementary tests or as the result of boolean procedures (functions) to be used in conditional instructions.

(ii) Primitive operators :

The main operations are the ones dealing with trees ; for a tree X :

- root(X) is the label of the root of X, thus an element of Ω.
- deg(X) is the arity of the root of X $\left(\text{i.e. } \delta(\text{root}(X))\right)$.
- for an integer i, X[i] denotes the i-th root subtree of X so that $X = \text{root}(X)\left(X[1],..,X[\deg(X)]\right)$.

Elementary tests allow comparison of the root of a tree to some element in Ω and of its degree to some fixed integer.

Results of tests and boolean functions can also be combined using the standard boolean connectives.

(iii) The presentation of the syntax is Pascal-oriented.

Procedure and program definitions allow usual sequential composition of instructions denoted by ";".

Procedure calls are written as usual : A $(Y_1,..,Y_m)$ denotes application of procedure A to arguments $Y_1, Y_2, \ldots Y_m$, with arguments being passed by reference.

Procedure definitions obey the format :

procedure A $(X_1 : T_1 ; X_2 : T_2 ;\ldots)$: ...endproc ;

function A $(X_1 : T_1 ; X_2 : T_2 ;\ldots)$: ...endfct ;

in the case of a pure procedure without result and of a boolean function respectively.

The control structures are :

conditionals :

if < boolean expression > then < instruction > else < instruction > fi .

iteration :

with Q a predicate over $\mathbb{N}^m \times \Omega^m$ it reads :

for $(i_1,..,i_m)$ with $Q(i_1,...,i_m,root(X_1),...,root(X_m))$ do

$B(Y_1,...,Y_p,X_1[i_1],...,X_m[i_m])$ od.

This construct means that the body of the for-loop will be executed for all values of indices i , $1 \leq j \leq m$ and $1 \leq i \leq deg(X_j)$, which satisfy predicate Q, where Q may depend on the root labels of its arguments.

conditional iteration :

with arguments $X_1,X_2,...,X_m$ satisfying $deg(X_1) = deg(X_2) =...= deg(X_m)$, it reads :

for i 1 to $deg(X_1)$ while Q $(X_1[i],...,X_m[i])$

do B $(X_1[i]...,X_m[i])$ od

With this mechanism, root-subtrees of a m-tuples of trees can be searched until a condition terminating the loop is met.

Notice that, in these iterations, the values of the indices are undefined outside the loop.

(iv) Special features :

These are the write, assign and nil constructs :

- write (<info>) : where info is a string possibly made of elements of Ω, can be used to transfer information on a write-only output file ; in particular, this feature allows programs to operate as tree transducers, the results being output for instance in polish prefix notation (trivial modifications would make it possible to produce linked tree-structures as outputs).
- assign (<boolean expression>) : each boolean function is assumed to have a special result register ; the assign construct can be pu around any boolean expression in a program definition and will result in assigning the value of the expression to the result register of the function that commands it.
- nil : this dummy procedure has no effect on the computation and will be used for convenience.

(v) macroinstructions :

It is convenient to avoid long sequences of nested conditional statements ; we shall therefore use in some cases the following macroinstruction :

```
case    root(X) of
 ω1  : <instruction>
 ω2  : <instruction>
  .
  .
  .
fo
```

which is easily translated in terms of if then else. For complexity estimates, we shall consider that this switch performs a single test (this convention could clearly be changed without deeply affecting our results).

In Figure 2, we give an example of program in PL-tree, which tests two trees for equality, using a recursive search in preorder.

```
function   equal (X,Y : T) :
if root(X) ≠ root(Y)   then assign (false)
else       assign (true) ;
           for  i← 1 to  deg(X)
           while assign (equal (X[i],Y[i]))

           do  nil  od
   fi
end equal .
```

Figure 2 : The function equal which tests two trees for equality.

2.2. - Complexity Rules

We define here input descriptors and complexity descriptors associated to PL-tree data structures and programs. We later give a set of rules that can be used to inductively determine the complexity descriptors of programs and procedures from their simpler components.

Let $U \subset T^m$ be a set of m-tuples of trees.

The generating function -or characteristic function- of U is the series

$$\chi u\ (x_1, x_2, \ldots, x_m) = \sum_{n_1, n_2, \ldots, n_m} c_{n_1, n_2, \ldots n_m}\ x_1^{n_1}\ x_2^{n_2} \ldots x_m^{n_m} \qquad (3)$$

where

$$c_{m_1, n_2, \ldots, n_m} = \text{card}\ \left\{ (X_1, X_2, \ldots, X_m) \in U\ /\ |X_1| = n_1, \ldots, |X_m| = n_m \right\}. \qquad (4)$$

Let furthermore Q be a predicate over T^m ; we also introduce the conditional characteristic function of U with condition Q as :

$$\chi u\ (x_1, x_2, \ldots, x_m \mid Q) = \chi\ (U \wedge Q)\ (x_1, x_2, \ldots x_m), \qquad (5)$$

where $U \wedge Q$ ist the conjonction (intersection) of U and Q;

In the sequel, we repeatedly make use of the notation

$$\left[\ x_1^{n_1}\ \ x_2^{n_2}\ \ldots\ x_m^{n_m}\ \right]\ \ f(x_1, x_2, \ldots, x_m)$$

to denote the Taylor coefficient of $x_1^{n_1}\ x_2^{n_2} \ldots x_m^{n_m}$ in $f(x_1, x_2, \ldots, x_m)$. With this notation, we see that (5) is equivalent to :

$$\left[x_1^{n_1}\ x_2^{n_2} \ldots x_m^{n_m}\right]\ \chi u\ (x_1, x_2, \ldots, x_m \mid Q) =$$

$$\text{card}\ \left\{ (X_1, X_2, \ldots, X_m) \in U\ /\ |X_1| = n_1, \ldots, |X_m| = n_m\ ,\ Q(X_1, \ldots, X_m) \right\}$$

Similarly, let $A(X_1 : T_1, X_2 : T_2, \ldots, X_m : T_m)$ be a procedure defined on m arguments of respective types $T_1, T_2, \ldots T_m$. Assume also that a standard complexity measure τ is defined for programs in PL-tree, with

$$\tau A(X_1, X_2, \ldots, X_m)$$

representing the cost of running procedure A on arguments $X_1, X_2, \ldots, X_m$. The complexity descriptor of A is defined as the series

$$\tau a(x_1, x_2, \ldots x_m)$$

whose general coefficient is

$$\left[x_1^{n1}\, x_2^{n2} \ldots x_m^{nm} \right] \tau a(x_1, x_2, \ldots, x_m) = \sum_{|X_1|=n_1, \ldots, |X_m|=n_m} \tau A(X_1, X_2, \ldots, X_m) \qquad (6)$$

and represents the cumulative cost of running algorithm A over all inputs $(X_1, X_2, \ldots, X_n)$ of size $(n_1, n_2, \ldots n_m)$.

For Q a predicate on $T_1 \times T_2 \times \ldots \times T_m$, the conditional complexity descriptor of A (under condition Q) is similarly defined as the series

$$\tau a(x_1, x_2, \ldots x_m \mid Q)$$

with

$$\left[x_1^{n1}\, x_2^{n2} \ldots x_m^{nm} \right] \tau a(x_1, x_2, \ldots, x_m \mid Q) =$$

$$\sum_{\substack{|X_1|=n_1, \ldots, |X_m|=n_m \\ Q(X_1, \ldots, X_m)}} \tau A(X_1, X_2, \ldots, X_m).$$

Thus $\tau a(x_1, x_2, \ldots x_m \mid Q)$ describes the complexity of A when applied to arguments satisfying condition Q.

In the following we shall use extensively a vector notation to denote m-tuples of variables and cartesian products :

$\underset{\sim}{X}$, $\underset{\sim}{x}$ and $\underset{\sim}{n}$ will denote respectively

$(X_1, X_2, \ldots, X_m)$, $(x_1, x_2, \ldots, x_m)$ and $(n_1, n_2, \ldots, n_m)$.

$\underset{\sim}{T}$ will denote $T_1 \times T_2 \times \ldots \times T_m$.

In particular, we shall write $\underset{\sim}{X} \in \underset{\sim}{T}$ instead of $(X_1, X_2, \ldots, X_m) \in T_1 \times T_2 \times \ldots \times T_m$ and

$|\underset{\sim}{X}| = \underset{\sim}{n}$ instead of $|X_i| = n_i$ for $1 \leq i \leq m$.

It also proves convenient to use $\underset{\sim}{X}\,[\underset{\sim}{i}]$ to denote $(X_1[i_1], X_2[i_2],\ldots,X_m[i_m])$ and $\underset{\sim}{x}^{\underset{\sim}{n}}$ to denote $x_1^{n_1} x_2^{n_2} \ldots x_m^{n_m}$ (thus $\underset{\sim}{x}^{\underset{\sim}{1}} = x_1 x_2 \ldots x_m$) ; however, for the sake of notational simplicity, we shall often use $\underset{\sim}{x}$ instead of $\underset{\sim}{x}^{\underset{\sim}{1}}$ when no ambiguity arises, so that :

$$[\underset{\sim}{x}] \quad f(\underset{\sim}{x}) = [\underset{\sim}{x}^{\underset{\sim}{1}}]\, f(\underset{\sim}{x}).$$

From the definitions follow a few basic facts which we now list :

(i) If $Q(\underset{\sim}{X}) = Q_1(\underset{\sim}{X}) \vee Q_2(\underset{\sim}{X})$ where Q_1 and Q_2 are <u>disjoint</u> predicates (i.e. $Q_1 \wedge Q_2 \equiv$ false) :

$$\chi q(\underset{\sim}{x}) = \chi q_1(\underset{\sim}{x}) + \chi q_2(\underset{\sim}{x})$$

$$\tau a(\underset{\sim}{x} \mid Q) = \tau a(\underset{\sim}{x} \mid Q_1) + \tau a(\underset{\sim}{x} \mid Q_2). \qquad (8)$$

If $Q(\underset{\sim}{X}) = Q_1(X_1) \wedge Q_2(X_2) \wedge .. \wedge Q_m(X_m)$ then

$$\chi q(\underset{\sim}{x}) = \chi q_1(x_1) \cdot \chi q_2(x_2) \cdot \ldots \cdot \chi q_m(x_m) ,$$

with χtrue = 1 and χfalse = 0 (9)

(ii) Let $t(x)$, $t_i(x)$ denote the characteristic functions of types T, T_i (since they will be of constant use we omit there the prefix χ).

Let $\Phi(u)$ ($\Phi_i(u)$ respectively) be the power series defined from Ω (Ω_i resp.) by

$$[u^n]\,\Phi(u) = \mathrm{card}\left\{ \omega \in \Omega \mid \deg(\omega) = n \right\} , \qquad (10)$$

with a similar definition for the Φ_i's. Then $t(z)$ satisfies the equation ([MM78], [G65], [SF82], [FO82]) :

$$t(x) = x\,\Phi\Big(t(x)\Big) \qquad (11)$$

and similarly for t_i. From (9) follows in particular that the characteristic function of the type $T_1 \times T_2 \times .. \times T_m$ is

$$t_1(x)\, t_2(x) \ldots\ldots t_m(x) \quad .$$

(iii) If the procedure $A(\underset{\sim}{X} : \underset{\sim}{T})$ depends effectively on only p arguments where $p < m$ -say $X_1, X_2, \ldots, X_p$- , it can be written as :

$$A(X_1, \ldots, X_m) = B(X_1, \ldots, X_p)$$

and we have

$$\tau a(x) = \tau b(x_1, \ldots, x_p) \cdot \prod_{p<i\leq m} t_i(x_i). \qquad (12)$$

This last equation is easily relativized to some predicate $Q_1 \times Q_2 \times \ldots \times Q_m$ over $T_1 \times T_2 \times \ldots \times T_m$:

$$\tau a(\underset{\sim}{x} \mid Q_1 \times Q_2 \times \ldots \times Q_m) = \tau b(\underset{\sim}{x} \mid Q_1 \times \ldots \times Q_p) \cdot \prod_{p<i\leq m} \chi q_i(x_i) \qquad (12')$$

(iv) Given a procedure $A\ (\underset{\sim}{X} : \underset{\sim}{T})$ and its complexity descriptor $\tau a(\underset{\sim}{x})$, the series $\tau a(z,z,\ldots,z)$ is often written for short as $\tau a(z)$. Its n-th Taylor coefficient satisfies

$$[z^n]\ \tau a(z) = \sum_{|X_1|+\ldots+|X_m|=n} \tau A(X_1,\ldots,X_m) \qquad (13)$$

which therefore represents the cumulated cost of procedure A on all m-tuples of inputs of total size n.

We can now proceed with the statement of complexity rules. Each rule applies to a construct in the language of the form

$$A = C\ (B_1, B_2, \ldots)$$

and expresses the complexity descriptor τa of A in terms of the complexity descriptors τb_i of the program segments B_i and the characteristic functions of the underlying datatypes :

$$\tau a(\underset{\sim}{x}) = \Gamma\left(\tau b_1(\underset{\sim}{x}), \tau b_2(\underset{\sim}{x}), \ldots, \underset{\sim}{t}(\underset{\sim}{x})\right)$$

Occasionally (in the case of boolean constructs), Γ may also involve the characteristic functions of some intervening boolean procedures.

Rules are relative to an additive (w.r.t. composition of instructions) complexity measure corresponding to execution time on an abstract machine model, whose properties are summarized in Figure 3. There is naturally considerable arbitrariness in the choices made in this table : our purpose has only been to have rules that are reasonably simple to state ; from the developments that follow it should be clear that adequate time constants in procedure calls, tests, branching... could be introduced to reflect more closely the time constants of any particular machine model.

Construct	Resulting complexity
$A \equiv B ; C$	$\tau A = \tau B + \tau C$
$A \equiv B(X[i])$	$\tau A = \tau B\,(X[i])$
$A \equiv$ <u>if</u> Q <u>then</u> B <u>else</u> C <u>fi</u>	$\begin{cases} \tau A = \tau Q + \tau B & \text{if } Q \\ \tau A = \tau Q + \tau C & \text{if } \neg Q \end{cases}$
$A \equiv$ <u>for</u> $\underset{\sim}{i}$ with Q <u>do</u> B <u>od</u>	τA is the sum of τB's corresponding to arguments which satisfy predicate Q and of τQ's.
$A \equiv$ for $\underset{\sim}{i}$ <u>while</u> Q <u>do</u> B <u>od</u>	τA is the sum of τQ's an τB's corresponding to executions of Q and B while condition Q is satisfied and, of the first execution of Q which returns false.
Q ≡ a boolean combination of atomic predicates on root labels and degrees	$\tau Q \equiv 1$
assign, nil	$\tau assign = \tau nil = 0$

<u>Figure 3</u> : A summary of the definitions of the additive measure considered.

Rule 1 [Composition]

When

$A(\underset{\sim}{X}) \equiv B(\underset{\sim}{X})\ ;\ C(\underset{\sim}{X})$

where $\underset{\sim}{X}$ is of type T, one has for any predicate Q on $\underset{\sim}{T}$:

$$\tau a(\underset{\sim}{x} \mid Q) = \tau b(\underset{\sim}{x} \mid Q) + \tau c(\underset{\sim}{x} \mid Q).$$

Rule 2 [Conditionals]

When

$A(\underset{\sim}{X}) \equiv$ if $Q(\underset{\sim}{X})$ then $B(\underset{\sim}{X})$ else $C(\underset{\sim}{X})$ fi

one has

$$\tau a(\underset{\sim}{x}) = \tau q(\underset{\sim}{x}) + \tau b\left(\underset{\sim}{x} \mid Q(\underset{\sim}{X})\right) + \tau c\left(\underset{\sim}{x} \mid \neg Q(\underset{\sim}{X})\right)$$

These two rules follow directly from the additive character of the time complexity measure.

Rule 3 [Subtree descent]

When

$A(X) \equiv B\left(X[i_1],\ldots,X[i_p]\right)$

for some i_j all different, where X is of type T, one has :

$$\tau a(x) = x\ \tau b\,(x,x,\ldots,x)\ \psi\left(t(x)\right)$$

where

$$\psi(u) = \sum_{\substack{\omega \in \Omega \\ p < \delta(\omega)}} u^{\delta(\omega)-p}$$

A variant of Rule 3 is of special interest when we insist that the rootsubtrees be identical ; we state it in a special case.

Let Eq_ω be the predicate over T defined by :

$$Eq_\omega\ (X) \equiv \left\{ root(X) = \omega \text{ and } \forall j \quad X[j] = X[1] \right\}.$$

We then have :

Rule 3 eq [Subtree descent with Equality]

When

$$A(X) \equiv B\Big(X[i]\Big)$$

where X is of type T and $i \leq \delta(\omega)$, one has under condition Eq_ω :

$$\tau a(x \mid Eq_\omega) = x\ \tau b\left(x^{\delta(\omega)}\right) \ .$$

Rule 4 [Iteration]

When

$$A(\underset{\sim}{X}) \equiv \underline{\text{for}}\ \ \underset{\sim}{i}\ \ \underline{\text{with}}\ \ Q\left(\underset{\sim}{i},\ \text{root}\ (\underset{\sim}{X})\right)\ \ \underline{\text{do}}\ \ B(\underset{\sim}{X}[\underset{\sim}{i}])\ \ \underline{\text{od}}$$

with $\underset{\sim}{X}$ of type $\underset{\sim}{T}$ one has [†]:

$$\tau a(\underset{\sim}{x}) \equiv \sum_{\underset{\sim}{1} \leq \underset{\sim}{i} \leq \underset{\sim}{m}} x_i t_i(x_i)\, \Phi_i'\left(t_i(x_i)\right) + \tau b(\underset{\sim}{x})\ \underset{\sim}{x}\ F_Q\left(t_1(x_1), \ldots, t_m(x_m)\right)$$

where

$$F_Q(\underset{\sim}{u}) = \sum_{\underset{\sim}{\delta}} a_Q(\underset{\sim}{\delta})\ \ \underset{\sim}{u}^{\underset{\sim}{\delta}-\underset{\sim}{1}}$$

and

$$a_Q(\underset{\sim}{\delta}) = \text{card}\left\{(\underset{\sim}{i},\underset{\sim}{\omega}) \mid \forall j\ \ \delta_j = \delta(\omega_j)\ \&\ Q(\underset{\sim}{i},\underset{\sim}{\omega})\right\} .$$

Two subcases of Rule 4 are of special interest so that we state them as derived rules :

Rule 4 dis [Distributive descent]

When

$$A(\underset{\sim}{X}) \equiv \underline{\text{for}}\ \ \underset{\sim}{i}\ \ \underline{\text{do}}\ \ B(\underset{\sim}{X}[i])\ \ \underline{\text{od}}$$

with $\underset{\sim}{X}$ of type $\underset{\sim}{T}$, one has :

$$\tau a(\underset{\sim}{x}) = \underset{\sim}{x}\ \tau b(\underset{\sim}{x}) \prod_{i \leq j \leq m} \Phi_j'\left(t_j(x_j)\right) .$$

---

† Φ', Φ_i' denote the derivatives of Φ and Φ_i.

Let Eqrt be the predicate over $\underset{\sim}{T}$ defined by

$$Eqrt(\underset{\sim}{X}) \equiv \forall i\ 1 \le i \le m \quad root(X_1) = root(X_i) \quad .$$

The second derived rule states as follows :

Rule 4 sim [simultanious descent]

When

$$A(\underset{\sim}{X}) \equiv \underline{for}\ \ i \leftarrow 1\ \ \underline{to}\ \deg\ (X_1)\ \ \underline{do}$$
$$B(X_1[i],\ X_2[i],\ldots,\ X_m[i])\ \ \underline{od}$$

where $\underset{\sim}{X}$ is of type T^mone has under condition Eqrt :

$$\tau a(\underset{\sim}{x} \mid Eqrt) = \underset{\sim}{x}\ \tau b(\underset{\sim}{x})\ \Phi'\left(\prod_{1 \le j \le m} t(x_j)\right) .$$

A variant of Rule 4 is also of interest when the descent affects a subset of the input arguments :

Rule 4 pds [Partial descent]

When

$$A(X,Y) \equiv \underline{for}\ i \leftarrow 1\ \text{to}\ \deg(Y)\ \ \underline{do}\ \ B(X,Y[i])\ \underline{od}$$

where Y is in T , one has

$$\tau a(x,y) = y\ \tau b(x,y)\ \Phi'\left(t(y)\right) .$$

Rule 5 [Conditional iteration]

When

$$A(\underset{\sim}{X}) \equiv \underline{for}\ \ i \leftarrow 1\ \ \underline{to}\ \deg\ (X_1)\ \underline{while}\ \ Q(\underset{\sim}{X}[i])$$
$$\underline{do}\ \ \ B(\underset{\sim}{X}[i])\ \ \underline{od}$$

where $\underset{\sim}{X}$ is in $\underset{\sim}{T}$, one has under condition Eqrt :

$$\tau a(\underset{\sim}{x} \mid Eqrt) = \underset{\sim}{x}\left(\tau q(\underset{\sim}{x}) + \tau b(\underset{\sim}{x})\right) \frac{F(\prod t_j(x_j) - F(\chi q(\underset{\sim}{x}))}{\prod t_j(x_j) - \chi q(\underset{\sim}{x})}$$

where

$$F(u) = \sum_{\omega \in I} u^{\delta(\omega)} \quad , \; I = \bigcap_j \Omega_j \; .$$

As a final rule, we give the complexity descriptor of the procedure that copies a tree on the outputfile :

Rule 6 [Copy]

With X in T one has :

$$\tau\text{copy}(x) = xt'(x).$$

This rule actually follows from the preceding ones as we shall sea later.

We now turn to equations for characteristic functions of boolean procedures. The basic remark is that the result of a boolean function is precisely that of the last "assign"-instruction executed ; the situation is therefore more intricate for characteristic functions than for complexity descriptors ; however some schemes are quite often encountered for which we give the following rules :

Rule 7 [Characteristic functions]

(i) When

$$a(\underset{\sim}{X}) \equiv \underline{\text{if}}\ Q(\underset{\sim}{X})\ \underline{\text{then}}\ \text{assign}\left(B(\underset{\sim}{X})\right)\ \underline{\text{else}}\ \text{assign}\left(C(\underset{\sim}{X})\right)\ \underline{\text{fi}}$$

where $\underset{\sim}{X}$ is of type $\underset{\sim}{T}$, one has :

$$\chi a(\underset{\sim}{x}) = \chi b(\underset{\sim}{x} \mid Q) + \chi c(\underset{\sim}{x} \mid \neg Q).$$

(ii) When

$$A(\underset{\sim}{X}) = \underline{\text{for}}\ i \leftarrow 1\ \underline{\text{to}}\ \deg(X_1)\ \underline{\text{while}}\ \text{assign}\left(Q(\underset{\sim}{X}[i])\right)\ \underline{\text{do}}\ B(\underset{\sim}{X}[i])\ \underline{\text{od}}$$

where $\underset{\sim}{X}$ is in $\underset{\sim}{T}$ and B is a pure procedure (without boolean result) one has under condition Eqrt :

$$\chi a(\underset{\sim}{x} \mid \text{Eqrt}) = \underset{\sim}{x}(F(\chi q(\underset{\sim}{x})) - F(0))$$

where

$$F(u) = \sum_{\omega \in I} u^{\delta(\omega)}, \; I = \bigcap \Omega_j \; .$$

This last assertion expresses the fact that predicate Q holds true for all m-tuples $\underset{\sim}{X}[i]$, $1 \leq i \leq \deg(X_1)$.

2.3. - Proof techniques

Counting multisets is the main tool for proving complexity rules. To introduce the method let us start with the problem of counting trees in a family $\mathcal{T}$ defined by a set Ω of operators :

$$\mathcal{T} = \sum_{\omega \in \Omega} (\mathcal{T}, \mathcal{T}, \ldots, \mathcal{T}),$$

or in other words :

$$\mathcal{T} = \sum_{p \geq 0} \sum_{\substack{\omega \\ \delta(\omega)=p}} \sum_{(T_1,\ldots,T_p) \in \mathcal{T}^p} \omega(T_1,\ldots,T_p). \qquad (14)$$

The generating function $t(x)$ of $\mathcal{T}$ can also be written as :

$$t(x) = \sum_{T \in \mathcal{T}} x^{|T|} \quad ; \qquad (15)$$

so that it can be formally derived from the multiset $\mathcal{T}$ by replacing each node (or symbol) of a $T \in \mathcal{T}$ by an x, taking juxtaposition of variables as products.

Applying this transformation to equation (14) we thus obtain :

$$t(x) = \sum_{\omega} x\Big(t(x)\Big)^{\delta(\omega)}$$

$$= x \sum s_n \Big(t(z)\Big)^n \qquad (16)$$

where s_n is the number of operators of arity n in Ω.

This symbolic way of deriving generating functions is inspired by works of Schutzenberger [FS 70]. It can be extended to count multisets of trees as shown in [G65] [SF 82] and in parallel developments of [BR 82]. It makes it possible to translate "at sight" inductive definitions of multisets obtained by combinations of sums and products into equations over generating functions. Extensive use of it is made in the sequel, and we shall illustrate it by means of a few examples.

Given a procedure $A(X : \underset{\sim}{T})$ we consider the multiset associated to the cost measure :

$$\tau A = \sum_{\underset{\sim}{X} \in \underset{\sim}{T}} \tau A(\underset{\sim}{X}) \cdot \underset{\sim}{X} \quad . \tag{17}$$

(a) The proof of Rule 3 [subtree descent] can be derived as follows ; by definition we have on multisets :

$$\begin{aligned} \tau A &= \sum_{X \in T} \tau B(X[i_1], \ldots, X[i_p]) \cdot X \\ &= \sum_{\delta(\omega) \geq p}^{\omega} \sum_{X_1, \ldots, X_{\delta(\omega)} \in T} \tau B(X_1, \ldots, X_{i_p}) \cdot \omega(X_1, \ldots, X_{\delta(\omega)}) \\ &= \sum_{\delta(\omega) \geq p}^{\omega} \omega \left\{ \tau B \, , \, T^{\delta(\omega)-p} \right\} \end{aligned} \tag{19}$$

where the brackets indicate the arguments of τB should be distributed in the positions i_1 to i_p. Applying the translation scheme on (19) we obtain :

$$\tau a(x) = \sum_{\delta(\omega) \geq p}^{\omega} x \, \tau b(x, \ldots, x) \left(t(x) \right)^{\delta(\omega)-p} \tag{20}$$

from which the rule follows.

(b) The proofs of rules 1-2-4-5-7 are quite similar, although multivariate series are used ; consider for instance Rule 5 [Conditional iteration] ; by definition :

$$\tau A(\mid Eqrt) = \sum_{\omega} \Bigg\{ \sum_{\underset{\sim}{X}} \tau Q\,(\underset{\sim}{X}[1]) \;.\; \underset{\sim}{X} + \sum_{1<p\le\delta(\omega)} \sum_{\underset{\sim}{X}} \Big(\tau B(\underset{\sim}{X}[p-1]) + \tau Q(X[p]) \Big) \;.\; \underset{\sim}{X} + \sum_{\substack{\underset{\sim}{X} \\ \wedge Q(\underset{\sim}{X}[i]) \\ 1\le i\le\delta(\omega)}} \tau B\Big(\underset{\sim}{X}[\delta(\omega)]\Big) \;.\; \underset{\sim}{X} \Bigg\} \qquad (21)$$

where $\sum_{\underset{\sim}{X}}$ means that the summation is taken over the m-tuples
$\langle X_j[1], \ldots X_j[\delta(\omega)]\rangle \in T_j^{\delta(\omega)}$, $1\le j\le m$,
and $X_j = \omega\Big(X_j[1],\ldots,X_j[\delta(\omega)]\Big)$.

Applying the translation scheme, we associate to trees in T_j the variable x_j , $1\le j\le m$ and we obtain :

$$\tau a(\underset{\sim}{x}) = \sum_{\omega} x_1 \ldots x_m \sum_{1\le p\le\delta(\omega)} \chi q(\underset{\sim}{x})^{p-1} \Big(\tau b(\underset{\sim}{x} \mid Q) + \tau q(\underset{\sim}{x})\Big) \cdot \prod_{1\le j\le m} \Big(t_j\,(x_j)\Big)^{\delta(\omega)-p}$$

$$= \underset{\sim}{x} \Big(\tau b(\underset{\sim}{x} \mid Q) + \tau q(\underset{\sim}{x})\Big) \sum_{\omega} \sum_{p} \chi q(\underset{\sim}{x})^{p-1} \Big(\prod t_j(x_j)\Big)^{\delta(\omega)-p} \;. \qquad (22)$$

The last sum being a geometric progression, we obtain the final form of the rule.

Rule 6 could be also proved along these lines but we can obtain it through the other rules by expressing the procedure copy recursively as :

```
procedure   copy (X : T) :
        write (root(X)) ;
        for  i ← 1  to deg(X)  do  copy (X[i])  od
end copy.
```

Then by Rules 1 and 4 we have

$$\tau copy\ (x) = t(x) + x\ \tau copy(x)\,\Phi'\Big(t(x)\Big)$$

solving this linear equation and reducting the expression we obtain :

$$\tau copy\ (x) = x t'(x) \;. \qquad (23)$$

2.4. - Analytic translation methods

It should be clear at this stage, that given a program (viz. a set of procedures) in PL-tree, one is able to derive, using the preceding rules, a system of equations which determines various generating functions associated to the program and which ultimately define the complexity descriptor. In this paragraph, we are interested in determining the behaviour of the average cost, asymptotically when the size of the data gets large.

Most complexity descriptors associated to simple programs satisfy a system of equations of the form :

$$\left\{ f_i\left(\underset{\sim}{x}\right) = H_i\left(\underset{\sim}{x}, \underset{\sim}{f}\left(\underset{\sim}{h}(\underset{\sim}{x})\right)\right), \quad 1 \le i \le p \right. \tag{24}$$

Where the h's and H's are known functions; this is usually achieved after some algebraic transformations, by eliminating conditions from conditional complexity descriptors.

A special case of interest is when the system can be written in the form

$$\left\{ f_i\left(\underset{\sim}{x}\right) = x_i \, H_i\left(\underset{\sim}{f}(\underset{\sim}{x})\right) \right. \tag{25}$$

It is then possible to solve it algebraically and to recover the coefficients of the Taylor expansions of the f's using the Lagrange-Good inversion theorem for implicit functions [G 60] ; however the expressions thus obtained are usually too complex to allow aymptotic analysis. We therefore turn to a more direct approach that follows the lines of [MM 78], [O 82], [SF 82], [FO 82] : this approach is based on the existence of relations between the asymptotics of Taylor coefficients of a function and the behaviour of that function around its main singularity.

More precisely, the cumulated costs relative to total input size n have a generating function which is obtained by replacing in (24) all occurrences of x_i's by z ; so that the system (24) becomes :

$$\left\{ g_i(z) = K_i\left(z, \underline{g}\left(\underline{k}(z)\right)\right) \quad 1 \le i \le p \right. . \tag{26}$$

In (26) one of the functions $g_i(z)$, call it c(z), is the complexity descriptor of the algorithm considered ; in order to determine the asymptotic behaviour of the coefficients c_n of c(z) we study its singularities. The method relies on two remarks :

(i) let ρ be the radius of convergence (r.d.c.) of c(z) ; then the exponential order of growth of c_n is ρ^{-n} , that is

$$\forall \varepsilon > 0 \quad [\rho^{-1}(1+\varepsilon)]^{-n} \underset{i.o}{<} c_n \underset{a.e}{<} [\rho^{-1}(1+\varepsilon)]^{-n}$$

(ii) If $z=\rho$ is the unique singularity of c(z) of modulus ρ, the behaviour of

$r_n = c_n \rho^n$ is determined by the local expansion of c(z) at $z=\rho$.

The major fact concerning algorithms written in PL-tree is that in most cases c(z) behaves as $\left(1 - \frac{z}{\rho}\right)^{-\alpha}$; more precisely, at $z=\rho$, c(z) can be written :

$$c(z) = \left(1 - \frac{z}{\rho}\right)^{-\alpha} u(z) + v(z) \tag{27}$$

where u and v are regular at $z=\rho$, and $\alpha \in \mathbb{R} \setminus \{0,-1,-2\ldots\}$.

Property (27) allows us to state, by the Darboux theorem [H 74], [D 1878] that :

$$c_n = -u(\rho)\, \rho^{-n}\, \frac{n^{\alpha-1}}{\Gamma(\alpha)} \left(1 + 0\left(\frac{1}{n}\right)\right) \tag{28}$$

where Γ is the Euler gamma function.

Let us illustrate this method by deriving the asymptotic number of term trees defined on a fixed set of operators Ω. Let $\Phi(u)$ be the power series such that $s_n = [u^n]\,\Phi(u) = \text{card}\left\{\omega \in \Omega \mid \delta(\omega) = n\right\}$.

We know by (11) that the generating function $t(z)$ satisfies :

$$t(z) = z\,\Phi\Big(t(z)\Big).$$

Let us further assume that the conditions (C) below hold :

(i) the s_n are bounded by some fixed M

(ii) $\gcd\left\{ n \mid s_n \neq 0 \right\} = 1$

(Condition (ii) can easily be dispensed with).

Summarizing extensive discussions of [MM 78][SF 82] we have :

(a) $t(z)$ is analytic for all z, $|z| \leq \rho$, except $z=\rho$ where

$$\rho = \frac{1}{\Phi'(\tau)} < 1 \qquad (29)$$

and τ is the root of smallest modulus of the equation :

$$\tau\,\Phi'(\tau) = \Phi(\tau) \quad ; \qquad (30)$$

τ is always a real positive number

(b) at $z=\rho$, $t(z)$ has an expansion of the form :

$$t(z) = \tau + \gamma\left(1 - \frac{z}{\rho}\right)^{+1/2} + \sum_{n>1} e_n \left(1 - \frac{z}{\rho}\right)^{n/2} \qquad (31)$$

where $\gamma = -\sqrt{\dfrac{2\Phi(\tau)}{\Phi''(\tau)}}$.

Similarly, the enumerating series for the family $\mathcal{T}^k$ of k-tuples of trees in $\mathcal{T}$ is $t^k(z)$, whose radius of convergence is still ρ, and at $z=\rho$ satisfies :

$$t^k(z) = \left(1 - \frac{z}{\rho}\right)^{1/2} u_k(z) + v_k(z) \qquad (32)$$

where u_k and v_k are regular and

$$u_k(\rho) = -\,k\,\tau^{k-1}\sqrt{\frac{2\Phi(\tau)}{\Phi''(\tau)}}$$

From the above discussion follows that the conditions for the Darboux theorem to apply are satisfied and we can state :

Proposition 1 : Let $\mathcal{T}$ be a family of trees satisfying conditions (C). Let $t_n \left(t_n^{(k)}\right)$ be the number of trees (k-tuples of trees) of size n ; then :

$$t_n = \sqrt{\frac{\Phi(\tau)}{2\pi\Phi''(\tau)}} \quad \rho^{-n} n^{-3/2} \left(1+0\left(\frac{1}{n}\right)\right)$$

$$t_n^{(k)} = k\tau^{k-1} \sqrt{\frac{\Phi(\tau)}{2\pi\Phi''(\tau)}} \quad \rho^{-n} n^{-3/2} \left(1+0\left(\frac{1}{n}\right)\right)$$

where $\rho = \frac{1}{\Phi'(\tau)}$ and $\tau\Phi'(\tau) = \Phi(\tau)$.

Local expansions of input and program descriptors around their singularities will be systematically used to derive the asymptotic behaviour of procedure costs, as given by the complexity rules of § 2.2. Let us give one example of this situation (we continue to assume here that the family of inputs $\mathcal{T}$ satisfies conditions (C))

Proposition 2 : Let B be a procedure on trees of type $\mathcal{T}$ and A be the iterate of B on subtrees, defined by :

$$A(X) \equiv B(X) \; ; \; \underline{\text{for}}\; i \leftarrow 1 \; \underline{\text{to}}\; \deg(X) \; \underline{\text{do}}\; A(X[i]) \; \underline{\text{od}}$$

Assuming $\tau b(z)$ has a unique algebraic singularity † on its circle of convergence, and

$$b_n \sim c.n^{\alpha}$$

for some constants c and $\alpha \leq \frac{1}{2}$ then

$$\overline{\tau a}_n \sim c\theta \; \frac{\Gamma\left(\alpha-\frac{1}{2}\right)^2}{\Gamma(\alpha)} \; n^{\alpha+\frac{1}{2}}$$

† A singularity is said algebraic whenever the function has a local expansion of the form (27).

where θ is a constant depending only on T, and Γ is the Euler gamma function.

Proof : By Rules 1 and 4dis we have :

$$\tau a(z) = \tau b(z) + z\, \tau a(z)\ \Phi'(t(z)),$$

hence

$$a(z) = \frac{\tau b(z)}{1 - z\,\Phi'(t(z))} \tag{33}$$

From equation (11) we get by derivation :

$$1 - z\Phi'(t(z)) = \frac{t(z)}{zt'(z)} \tag{34}$$

Since $\overline{\tau b_n}$ is equivalent to cn^{α}, $\tau b(z)$ has radius of convergence ρ, and from the hypothesis admits at $z=\rho$ (its singularity) the expansion :

$$\tau b(z) = \left(1 - \frac{z}{\rho}\right)^{-\beta} u_1(z) + v_1(z) \tag{35}$$

where u_1 and v_1 are regular at $z=\rho$.

Rewriting (31) as :

$$t(z) = (1 - \tfrac{z}{\rho})^{1/2}\ u(z) + v(z),$$

we obtain the expansion for $t'(z)$ at $z=\rho$:

$$t'(z) = -\frac{1}{2\rho}\left(1 - \frac{z}{\rho}\right)^{-1/2}\left[u(z) - 2\rho\left(1 - \frac{z}{\rho}\right) u'(z)\right] + v'(z). \tag{36}$$

Then $\tau a(z)$ has radius of convergence ρ and at $z=\rho$ we conclude from (33) (34) (35) (36) that it satisfies :

$$\tau a(z) = \left(1 - \frac{z}{\rho}\right)^{-\delta} u_2(z) + v_2(z) \tag{37}$$

where u_2 and v_2 are regular and

$$\delta = \begin{cases} \frac{1}{2} & \text{if } \beta \le 0 \\ \beta + \frac{1}{2} & \text{otherwise} \end{cases}.$$

Furthermore when $\beta > 0$

$$u_2(\rho) = -u_1(\rho)\ \frac{\gamma}{2\tau}$$

where γ and τ are defined in equations (30) and (31).

By proposition 1 and the Darboux theorem it is easy to see that

$$\alpha = \beta + \frac{1}{2} .$$

Applying again Darboux' theorem to (37) leads to the asymptotic value of the average cost of procedure A. ■

3. DIFFERENTIATION ALGORITHMS

In this section we study a class of algorithms which perform formal differentiation on expressions represented by trees ; in a more algebraic setting these algorithms can also be seen as top-down finite state transducers [Th 73] and illustrate a natural class of computations on trees. Let us start with some definitions.

Let Ω be an arbitrary finite set of operators (including possibly $+, -, \sqrt{\ }, \log, \sin, \ldots$), variables $(x,y\ldots)$ and constants $(a,b,c,\ldots)$. A set of differentiation rules D over the family T of terms constructed on Ω is defined by local transformations on terms : a term

$$X = \omega\,(X[1],\ldots,X[m])$$

with leading operator ω of arity m is transformed into a term

$$DX = \lambda\,(X[i_1],\ldots,X[i_p],\ DX[1],\ldots,DX[m])$$

where as usual the $X[i_j]$'s are root subtrees of X and $DX[i]$ denotes the derivative of $X[i]$ according to the set of derivation rules. In the derived expression, λ will be called the <u>header</u> ; let ε_i, δ_i , $i \geq 1$ be nullary symbols not in Ω ; the header λ of DX is a tree over

$$\Omega' = \Omega \cup \{\varepsilon_i \mid 1 \leq i \leq m\} \cup \{\delta_i \mid 1 \leq i \leq m\}$$

where δ_i's <u>occur exactly once</u> : the term DX is thus obtained by substituting $X[i]$'s to ε_i's and $DX[i]$'s to δ_i's. The fact that δ_i's occur exactly once is an important feature of differentiation ; thus in DX, derived subexpressions $DX[i]$, $1 \leq i \leq m$, occur once, whereas copied subexpressions $X[i]$'s, $1 \leq i \leq m$, may occur several times.

<u>Notations</u> : Let $\omega \in \Omega$ be an operator, with arity $\delta(\omega) = m$.
Let $\lambda\,(\varepsilon_1,\ldots,\varepsilon_m,\delta_1,\ldots,\delta_m)$ be the header of $D\left(\omega(X_1,\ldots,X_m)\right)$
The size $h(\omega)$ of the header λ associated to ω is the number of nodes of λ belonging to Ω (ε_i's and δ_i's are not counted). For $1 \leq i \leq m$, $\alpha(\omega)_i$ is the number of times subexpression $X[i]$ is copied (substituted to ε_i) in the derived expression , i.e. the number of occurrences of ε_i in λ ; the total number $\alpha(\omega)$ of copied subexpressions is $\alpha(\omega) = \alpha(\omega)_1 + \ldots + \alpha(\omega)_m$.

Figure 4 gives an example of differentiation rules.

Multiplication :

D* (X , Y) = + (* (X , DY) , * (DX , Y));

The header is + (* (. , .) , * (. , .)) so that
h(*) = 3 , α(*) = 1+1 = 2

Square root :

D $\sqrt{}$(X) = ÷ (DX , * (2 , $\sqrt{}$(X))) ;

the header is ÷ (. , *(2 , $\sqrt{}$(.))) so that
h($\sqrt{}$) = 5 , α($\sqrt{}$) = 1

Inverse :

D inv(X) = op (÷(DX , *(X , X))) ;
the header is op (÷(. ,*(. ,.))) so that
h(inv) = 3 α(inv) = 2

Figure 4 : Some usual differentiation rules.
(εi's and δi's have been marked by dots in headers).

The rules of some differentiation system D translate at sight into an algorithm written in PL-tree applied to inputs of type T , where $T=\Omega(T)$(see,e.g.,[Kn 68] p. 338). The general form of the algorithm is given in Figure 5 a ; the sequence of instructions corresponding to each branch of the case should be macroexpanded as is shown in Figure 5-b on an example.

```
procedure  diff (X : T) =

case  root(X) of

ω1  :  generate  λω1(X[i1],..,X[ip],diff(X[1]),.., diff(X[p]))

ω2  :  generate  λω2(.....)
.
:
ωn  : generate   λωn(......)
esac
end diff .
```

Figure 5-a : General form of the differentiation algorithm

```
*  :   write ('+') ; write ('*') ; copy (X[1]) ;
       diff (X[2]) ; write ('*') ; diff (X[1]) ;
       copy (X[2]).

inv :  write ('op') ; write ('÷' ) ; diff (X[1]) ;
      write ('*') ; copy (X[1]) ; copy (X[1]).
```

Figure 5-b : Macroexpansion of generate for two usual operators.

The complexity descriptor of procedure diff is therefore by Rule 2 [conditionals]

$$\tau diff(z) = t(z) + \sum_{\omega} \tau gener(z \mid root(X) = \omega) \qquad (38)$$

By Rules 1-3 and 6 one gets :

$$\begin{aligned} \tau gener(z \mid root(X)=\omega) = {} & h(\omega)\, z(t(z))^{\delta(\omega)} \\ & + \alpha(\omega)\, z^2 t'(z)\, (t(z))^{\delta(\omega)-1} \\ & + \delta(\omega)\, z\, \tau diff(z)(t(z))^{\delta(\omega)-1} \end{aligned} \qquad (39)$$

where the first term corresponds to write instructions, the second to copy instructions and the third one to recursive calls with subtree descent.

Combining (38) and (39) we finally get that :

$$\tau\text{diff}(z) = t(z) + zH(t(z)) + z^2 t'(z)\, A(t(z)) + z\tau\text{diff}(z)\ \Phi'(t(z)) \qquad (40)$$

where $H(u) = \sum_{\omega} h_\omega\, u^{\delta(\omega)}$

and $A(u) = \sum_{\omega} \alpha_\omega\, u^{\delta(\omega)-1}$.

Solving this linear equation in $\tau\text{diff}(z)$ we obtain the explicit form :

$$\tau\text{diff}(z) = \frac{t(z) + z\,H(t(z)) + z^2 t'(z)\, A(t(z))}{1 - z\Phi'(t(z))} \qquad (41)$$

and by (35)

$$\tau\text{diff}(z) = t'(z) + \frac{zt'(z)}{t(z)}\, H(t(z)) + z\, t'^2(z)\, \frac{A(t(z))}{t(z)} \qquad (42)$$

Since by the Lagrange inversion theorem we have explicit expressions for the coefficients of $t(z)$ and $t'(z)$ it would be possible to provide explicit but akward expressions for the coefficient of τdiff ; the formulae obtained in this way are quite intricate so that it would be hard -if not impossible- to derive an asymptotic estimate for the average cost of the algorithm.
We thus turn to the analytic study of $\tau\text{diff}(z)$.

Since Ω is finite, $H(u)$ and $A(u)$ are polynomials and one can conclude that $\tau\text{diff}(z)$ has radius of convergence ρ. Furthermore for $|z| = \rho$ its singularities are precisely those of $t(z)$ and $t'(z)$. We can thus obtain a local expansion of $\tau\text{diff}(z)$ at $z=\rho$ by simply replacing $t(z)$ and $t'(z)$ by $\tau + \gamma\left(1 - \frac{z}{\rho}\right)^{\frac{1}{2}}$ and $\tau' + \gamma'\left(1 - \frac{z}{\rho}\right)^{-\frac{1}{2}}$ respectively in equation (42) :

$$\tau\text{diff}(z) = \frac{\rho^2}{\tau}\,\alpha\,\gamma'^2\left(1 - \frac{z}{\rho}\right)^{-1} + \left(1 - \frac{z}{\rho}\right)^{-\frac{1}{2}} v(z) + w(z)$$

where $v(z)$ and $w(z)$ are analytic at $z=\rho$, and $\alpha = A(\tau)$.

Using the Darboux theorem we conclude that

$$\tau diff_n = [z^n]\, \tau diff(z) = \frac{\rho^2}{\tau}\, \alpha\, \gamma'^2\, \rho^{-n}\, (1 + O(n^{-1/2})).$$

From (28) we have for the average cost :

$$\overline{\tau diff_n} = \frac{\tau diff_n}{t_n} = \frac{\rho^2}{\tau}\, \alpha\, \gamma'^2 \sqrt{\frac{2\pi\Phi''(\tau)}{\Phi(\tau)}}\; n^{3/2} \left(1 + O\left(n^{-1/2}\right)\right)$$

Theorem 1 :

The average cost of a differentiation algorithm is of the form :

$$\tau diff_n = c\, n^{3/2} + O(n)$$

where c is a constant which depends only on the family of trees and the set of rules.

The behavior of the average cost ($O(n^{3/2})$) is to be compared to the worst case ($O(n^2)$) and the best case ($O(n)$). It is important to notice that the non linear cost is due to the copies of subtrees the algorithm performs. If we allow pointers so that common subexpressions could be shared, we could simply attach the subtrees of the argument to the headers with unit cost (independently of the size of the subtree).

Equation (41) would then be changed to

$$\tau\, diff(z) = t'(z) + \frac{z\, t'(z)}{t(z)} \left(H(t(z)) + A(t(z))\right) \qquad (43)$$

whose analysis yields a linear average cost consistent with the linear worst case behaviour.

4 - TREE COMPATIBILITY

In this section we study an algorithm which returns the greatest commun part of two trees in a family T ; this algorithm can appear as the first part of generalization algorithms. The interest relies now on the fact that the procedure has two arguments as one can see on Figure 6.

```
procedure compat (X , Y : T )
    if root(X) ≠ root(Y) then write ('.')
      else begin write (root(X)) ;
                for i ← 1 to deg(X) do compat (X[i], Y[i])
           end fi
end compat.
```

- Figure 6 -
Algorithm for tree-compatibility

The complexity descriptor of procedure compat satisfies the following equations ; by rule 2 we decompose τcompat in :

$$\tau\text{compat}(x,y) = t(x)\,t(y) + \tau\text{then}(x,y \mid \text{root}(X) \neq \text{root}(Y)) + \tau\text{else}(x,y \mid \text{root}(X) = \text{root}(Y)). \quad (44)$$

where the unit cost of the test in taken into account by the term $t(x)\,t(y)$.

Now

$$\tau\text{then}(x,y \mid \text{root}(X) \neq \text{root}(Y)) = t(x).t(y) - xy\,\Phi(t(x)\,t(y)) \quad (45)$$

since this quantity is the characteristic function of pairs $(X,Y) \in T^2$ whose roots are different.

By Rules 1 and 4sim one immediately obtains :

$$\tau\text{else}(x,y \mid \text{root}(X) = \text{root}(Y)) = xy\,\Phi(t(x)\,t(y)) + xy\,\tau\text{compat}(x,y)\;\Phi'(t(x)t(y)) . \quad (46)$$

Combining equations (44), (45), (46) and solving the linear equation in τcompat we get :

$$\tau\text{compat}(x,y) = \frac{2t(x)t(y)}{1 - xy\,\Phi'(t(x)t(y))} . \quad (47)$$

In order to derive the cost of procedure compat w.r.t. the total size of its arguments, we use remarks (11) and (24) and substitute variable z for x and y, obtaining (with an obvious change in notation) :

$$\tau\text{compat}(z) = \frac{2\,t^2(z)}{1 - z^2\Phi'(t^2(z))} \qquad (48)$$

It is easy to see that $\Phi'(t^2(z))$ is analytic for $|z| < \rho$, and that at $z=\rho$:

$$\Phi'(t^2(z)) = \Phi'(\tau^2) + 2\tau\gamma\Phi''(\tau^2)\left(1 - \frac{z}{\rho}\right)^{\frac{1}{2}} + O\left(1 - \frac{z}{\rho}\right)$$

Hence

$$\tau\text{compat}(z) = \tau_1 + \gamma_1\left(1 - \frac{z}{\rho}\right)^{\frac{1}{2}} + O\left(1 - \frac{z}{\rho}\right)$$

with

$$\tau_1 = \frac{2\,\tau^2}{1 - \rho^2\Phi'(\tau^2)} \quad \text{and} \quad \gamma_1 = \frac{4\tau\gamma(1-\rho^2\Phi'(\tau^2)) + 4\,\tau^5\rho^2\gamma\Phi''(\tau^2)}{(1-\rho^2\Phi'(\tau^2))^2} \,.$$

The average cost is therefore asymptotically constant :

$$\overline{\tau\text{compat}_n} = \frac{\gamma_1}{2\tau\gamma}\left(1 + O\left(\frac{1}{n}\right)\right) .$$

Theorem 2 :

For any family $\mathcal{T}$ of trees, there exists a real constant $c_{\mathcal{T}}$ such that the average cost of testing tree compatibility is

$$\text{compat}_n = c_{\mathcal{T}} + O\left(\frac{1}{n}\right) .$$

For many classes of trees this numerical constant can be easily computed ; for instance :

- for $\mathcal{G}$ the class of general planar trees whose generating function is defined with $\Phi(u) = \frac{1}{1-u}$ we have :

$$c_{\mathcal{G}} = \frac{39}{16}$$

- for $\mathcal{B}$ the class of binary trees ($\Phi(u) = 1+u^2$) we have

$$c_{\mathcal{B}} = 8 \,.$$

5 - TREE MATCHING AND SIMPLIFICATION

In this section we study two algorithms whose analysis are slightly more complex than the two previous ones. In the case of tree matching the algorithm consists of two procedures with two arguments taken in different families. In the case of simplification the equation to be solved for the cost generating function does not define it implicitly. Furthermore in each case, we have to handle characteristic series.

5.1. - Tree matching

The algorithm we study works as follows ; let P and T be the family of patterns and texts respectively. The class T is defined by a set Ω of operators ($T = \Omega(T)$). Let $\Omega^+ = \{\omega \in \Omega \mid \delta\ \omega > 0\}$ and # be a new symbol of arity 0. The class P of patterns satisfies the equation

$$P = [\Omega^+ \cup \{\#\}]\ (P) \quad .$$

A tree P in P is said to occur at the root of a tree T in P if there exist trees $T_1,\ldots,T_l$ in T which substituted to #'s in P give exactly T :

$$T = P\ (T_1,\ldots,T_l).$$

In a sense #'s play the role of a "don't care" which can be substituted by any tree in T.

More generally P is said to occur in T if it occurs at the root of some subtree of T.

The algorithm therefore tests for any subtree of T whether P occurs at its root ; this last procedure is a variant of procedure equal in Figure 2 ; the algorithm is described in Figure 7.

This algorithm has been extensively studied in [SF 82] ; we propose here a new proof of the main result -the expected linearity of the algorithm- in our framework.

```
function compare (P: P,T:T)  =
  if root (P) = #  then  assign (true)
     else if  root (P) ≠ root (T) then assign (false)
           else  for  i ← 1  to deg(P)
                 while assign (compare (P[i],T[i]))  do nil od
           fi  fi
end compare.
procedure match (P : P,T:T) =
  if compare (P,T) then write ('occurrence') fi
  for i ← 1  to deg(T)  do  match (P,T[i]) od
end  match.
```

Figure 7
The two procedures for tree-matching

Let p(x) and t(y) denote the generating functions associated to P and T respectively ; these series satisfy equations

$$\left\{\begin{array}{lll} & p(x) & = x(1 + \psi(p(x))) \\ & t(y) & = y\,\Phi(t(y)) \\ \text{where} & \psi(u) & = \Phi(u) - \Phi(0). \end{array}\right. \qquad (49)$$

We are now able to translate the algorithm in order to determine the complexity descriptor of procedure match ; we shall use variables x and y to represent patterns and texts respectively.

By Rules 1-2 and 4pds we have :

$$\begin{aligned} \tau\text{match}(x,y) &= \tau\text{comp}(x,y) + \chi\text{comp}(x,y) \\ &\quad + y\,\tau\text{match}(x,y)\;\Phi'(t(y)). \end{aligned} \qquad (50)$$

By Rules 2 and 5 we have :

$$\begin{aligned} \tau\text{comp}(x,y) &= p(x).t(y) + (p(x) - x)\,t(y) \\ &\quad + xy\,\tau\text{comp}(x,y)\,\frac{\psi(p(x)t(y)) - \psi(\chi\text{comp}(x,y))}{p(x)t(y) - \chi\text{comp}(x,y)}. \end{aligned} \qquad (51)$$

Finally we deduce from Rule 7 :

$$\chi\mathrm{comp}(x,y) = xt(y) + xy\ \psi(\chi\mathrm{comp}(x,y)). \qquad (52)$$

Following the standard approach we turn to univariate power series, expressing the cost w.r.t. the total size of the input ; equations (50), (51) and (52) are then rewritten as :

$$\tau\mathrm{match}(z) = \frac{\tau\mathrm{comp}(z) + \chi\mathrm{comp}(z)}{1 - z\,\Phi'(t(z))}\ , \qquad (50\text{ bis})$$

$$\tau\mathrm{comp}(z) = \frac{2\,p(z)\,t(z) - zt(z)}{1 - z^2\ \dfrac{\psi(p(z)t(z)) - \psi(\chi\mathrm{comp}(z))}{p(z)t(z) - \chi\mathrm{comp}(z)}} \qquad (51\text{ bis})$$

$$\tau\mathrm{comp}(z) = zt(z) + z^2\,\psi(\chi\mathrm{comp}(z)). \qquad (52\text{ bis}).$$

By remark (35), and by combining the last two equations we finally obtain the following system where the variable z is implicit :

$$\begin{cases} \tau\mathrm{match} = (\tau\mathrm{comp} + \chi\mathrm{comp})\,\dfrac{zt'}{t} \\ \tau\mathrm{comp} = \dfrac{(2pt - zt)\,(pt - \chi\mathrm{comp})}{pt - z^2\,\psi(pt) - zt} \\ \chi\mathrm{comp} = zt + z^2\,\psi(\chi\mathrm{comp}) \quad . \end{cases} \qquad (53)$$

We now turn to the analytic study of system (53) ; we assume, for simplicity, that conditions (C) are satisfied by T, hence by P.

Let as usual ρ be the radius of convergence of $t(z)$; at $z=\rho$, its dominant singularity we recall that :

$$\begin{cases} t(z) = \tau + \gamma\Delta + O(1 - \frac{z}{\rho}) \\ t'(z) = \gamma' \Delta^{-1} + \tau' + O(1 - \frac{z}{\rho}) \end{cases} \qquad (54)$$

where $\Delta = \sqrt{1 - \frac{z}{\rho}}$ and $\gamma' = -\frac{\gamma}{2\rho}$.

We first prove that $\chi comp(z)$ has its dominant singularity at $z = \rho$; since $[z^n]\chi comp(z) < [z^n]\, t^2(z)$, $\chi comp(z)$ has a radius of convergence $\geq \rho$ Furthermore by the implicit function theorem, $\chi comp(z)$ is uniquely defined and analytic unless

$$\frac{d\chi comp(z)}{dz} = \frac{t + zt' + 2z\ \psi(\chi comp)}{1 - z^2\ \psi'(\ comp)} \qquad (55)$$

becomes infinite.

Since for $|z| \leq \rho < 1$, $\chi comp(z) < |t(z)|$, and $1 \geq z\psi'(t(z)) > z^2\ \psi'(\chi comp(z))$ on the real axis, we conclude that the denominator in (55) cannot vanish, while the numerator becomes infinite at the only point $z=\rho$.

Using equation (54) we derive a local expansion for $\chi comp(z)$ at $z=\rho$, namely :

$$\chi comp(z) = \chi + \chi'\Delta + O(1 - \frac{z}{\rho}) \qquad (56)$$

where $\chi = \chi comp(\rho)$ is the least positive root of

$$\chi = \rho\tau + \rho^2\ \psi(\chi) ,$$

and $$\chi' = \frac{\rho\gamma}{1 - \rho^2\psi'(\chi)} . \qquad (57)$$

We now study $\tau comp(z)$ and $\tau match(z)$; since we know from the procedure structures that these algorithms are at most quadratic in the size of the input, their complexity descriptors should have the same radius of convergence as the characteristic series of the input , namely $p(z)t(z)$.

Let us then examine $p(z)$; two cases arise depending on the condition : $1 + \psi = \Phi$.

(i) $1 + \psi = \Phi$; then $p(z) = t(z)$; the dominant singularity of $\tau comp$ and $\tau match$ is easily seen to be located at $z=\rho$ and using (54) and (56) we conclude by elementary computations that at $z=\rho$ we have :

$$\begin{cases} \tau comp(z) = \xi + \xi'\Delta + O(1 - \frac{z}{\rho}) \\ \tau match(z) = \mu\Delta^{-1} + \mu' + O(|1 - \frac{z}{\rho}|^{1/2}) \end{cases} \tag{58}$$

where

$$\begin{cases} \xi = \dfrac{(2\tau^2 - \rho\tau)\,(\tau^2 - \chi)}{\tau^2 - \rho\tau - \rho^2\psi(\tau^2)} \\ \mu = -\dfrac{\gamma}{2\tau}\,(\xi + \chi) . \end{cases} \tag{59}$$

ii) $1 + \psi \neq \Phi$; it is then easy to see that $p(z)$ has a radius of convergence greater than ρ and is regular at $z=\rho$: let $p(\rho) = \bar{\omega}$; the dominant singularity of $\tau comp(z)$ and $\tau match(z)$ is again located at $z=\rho$ and by similar computations we conclude from (54) and (56) that equations (58) are still valid at $z=\rho$ with constants :

$$\begin{cases} \xi = \dfrac{(2\bar{\omega}\tau - \rho\tau)\,(\bar{\omega}\tau - \chi)}{\bar{\omega}\tau - \rho\tau - \rho^2\psi(\bar{\omega}\tau)} \\ \mu = -\dfrac{\gamma}{2\tau}\,(\xi + \chi) . \end{cases} \tag{60}$$

The characteristic function of the input is $p(z)t(z)$ which has radius of convergence ρ and a dominant singularity at $z=\rho$, where we have :

$$p(z)t(z) = \begin{cases} \tau^2 + \gamma\tau\Delta + O(1-\frac{z}{\rho}) & \text{when (i) holds} \\ \overline{\omega}\tau + \overline{\omega}\gamma\Delta + O(1-\frac{z}{\rho}) & \text{when (ii) holds} \end{cases} \qquad (61)$$

Therefore by the Darboux theorem we have :

$$\tau match_n = \frac{[z^n]\ \tau match(z)}{[z^n]\ p(z)t(z)} = \alpha n + O(1) \qquad (62)$$

where

$$\alpha = \begin{cases} \frac{\mu}{\gamma\tau} & \text{in case (i)} \\ \frac{2\mu}{\gamma\overline{\omega}} & \text{in case (ii)} \end{cases}$$

Theorem 3 :

The average running time of the maching algorithm is linear in the size of the input :

$$\tau match_n = \alpha n + O(1)$$

where the constant α depends only on the families of texts and patterns.

It is worth noticing that the worst case behaviour is quadratic in the size of the input.

5.2. - Algebraic simplification

As a last showcase we propose to study an algorithm which reduces arithmetic expressions containing subexpressions of the form $e_1 - e_2$, when e_1 and e_2 are formally identical. The algorithm is shown in Figure 8 and makes use of function equal described in Figure 2. The family A of expressions considered here is defined by a set Ω consisting of c variables or constants -e.g. 0,1,..., x,y,...- and d binary symbols -e.g. +,-,*,÷,... .

The generating functions associated to A satisfies therefore the equation

$$a(z) = cz + dza^2(z) \quad \text{or} \quad a(z) = z\Phi(a(z)) \qquad (63)$$

with $\Phi(u) = c + du^2$.

```
procedure  reduce (X : A) =
     if  root(X)  ≠  '-'  then   write (root(X)) ;
                                 reduce (X[1]) ;
                                 reduce (X[2])
     else  if  equal (X[1],X[2]) then write ('0')
                                 else write ('-') ;
                                      reduce (X[1]) ;
                                      reduce (X[2])
     fi   fi
end  reduce.
```

- Figure 8 -
The procedure for algebraic simplification.

By Rule 2 we can write with obvious notations :

$$\tau\text{reduce}(z) = a(z) \; + \tau\text{then}\,(z \mid \text{root}(X) \neq \text{'-'}) \; + \tau\text{else}\,(z \mid \text{root}(X) = \text{'-'})\,. \tag{64}$$

By Rules 1 and 3 we also have

$$\tau\text{then}\,(z \mid \text{root}(X) \neq \text{'-'}) = cz + (d-1)\, z\, a^2(z) + 2z(d-1)\, a(z)\, \tau\text{reduce}(z). \tag{65}$$

Finally, let

$$R(X) \equiv \text{reduce}\,(X[1])\;;\;\text{reduce}\,(X[2])\;;$$

by Rule 2 it is clear that :

$$\tau r(z) = \tau r(z \mid \text{equal}\,(X[1], X[2])) + \tau r(z \mid \neg\,\text{equal}\,(X[1], X[2]))\,,$$

hence by Rules 1-3 and 3 eq

$$\tau r(z \mid \neg\text{equal}\,(X[1], X[2])) = 2z\, a(z)\, \tau\text{reduce}(z) - 2z\, \tau\text{reduce}(z^2). \tag{66}$$

Therefore

$$\tau\text{else}\ (z \mid \text{root}(X) = '-') = \tau\text{eq}(z) + z a^2(z) + 2z a(z)\ \tau\text{reduce}\ (z) - 2z\ \tau\text{reduce}\ (z^2)\ , \qquad (67)$$

where

$$\tau\text{eq}(z) = z\ \tau\text{equal}\ (z,z)\ .$$

The study of τequal (x,y) is similar to that of τcomp (x,y) ; τequal is defined by the system :

$$\begin{cases} \chi\text{equal}(x,y) = xy\,\Phi\,(\chi\text{equal}(x,y)) \\ \tau\text{equal}(x,y) = a(x)a(y) + xy\ \tau\text{equal}(x,y)\ \dfrac{\Phi(a(x)a(y)) - \Phi(\chi\text{equal}(x,y))}{a(x)a(y) - \chi\text{equal}(x,y)} \end{cases}$$

from which we deduce that

$$\begin{cases} \chi\text{equal}\ (x,y) = a(xy) \\ \tau\text{equal}\ (x,y) = \dfrac{a(x)a(y)}{1 - dxy\ (a(x)a(y) + a(xy))} \end{cases} . \qquad (68)$$

Substituting z for x and y in equation (68) we get :

$$\tau\text{eq}(z) = z\ \frac{a^2(z)}{1 - dz^2\ (a^2(z) + a(z^2))} \ . \qquad (69)$$

Combining equations (64), (65), (67) and (35) we finally obtain :

$$\tau\text{reduce}(z) = [2\ a(z) + \text{eq}(z) - 2z\ \ \text{reduce}\ (z^2)]\ \frac{za'(z)}{a(z)} \ . \qquad (70)$$

We now turn to the analytic study of a(z), τeq(z) and τreduce (z).

From equation (63) we find :

$$a(z) = \frac{1 - \sqrt{1 - 4cdz^2}}{2dz} \quad ; \qquad (71)$$

therefore a(z) has radius of convergence $\rho = \frac{1}{2\sqrt{cd}}$, and two singularities in $z = \pm\rho$ where it takes values $\pm\tau$, with $\tau = \frac{c}{d}$.

Its derivative a'(z) and the function $\frac{za'(z)}{a(z)}$ have similarly ρ as radius of convergence and two singularities in $z = \pm\rho$, since they are expressed as :

$$a'(z) = -\frac{1}{2dz^2}\left(1 - \sqrt{1 - 4cdz^2}\right) + \frac{2c}{\sqrt{1 - 4cdz^2}}$$

and $$\frac{za'(z)}{a(z)} = \frac{1}{\sqrt{1 - 4cdz^2}} \quad . \qquad (72)$$

Considering worst case behaviors of procedures reduce and equal we deduce that for some real constant λ :

$$[z^n]\ a(z) \le [z^n]\ \tau eq(z) \le \lambda n [z^n]\ a(z) \qquad \text{and}$$

$$[z^n]\ a(z) \le [z^n]\ \tau reduce(z) \le \lambda n^2\ [z^n]\ a(z) \ .$$

Thus $\tau eq(z)$ and $\tau reduce(z)$ have radius of convergence ρ . Since $\rho < 1$, functions $a(z^2)$ and $\tau reduce(z^2)$ are regular in a domain that strictly contains $\{ z \mid |z| \le \rho \}$ (see [Po 37] for a similar argument). Futhermore the denominator of $\tau eq(z)$ does not vanish for $|z| \le \rho$ the series $h(z) = dz^2(a^2(z) + a(z^2))$ being monotonically increasing on the real positive axis and $h(\rho) < 1$. Therefore $\tau eq(z)$ has only two singularities for $|z| \le \rho$ at $z = \pm\rho$ and the same holds true for $\tau reduce(z)$.

From equations (69), (70), (71) and (72) we derive by elementary algebraic manipulations, a local expansion of $\tau\text{reduce}(z)$ at $z=\rho$ in terms of $\sqrt{1 - \frac{x}{\rho}}$:

$$\tau\text{reduce}(z) = u(z)\frac{1}{\sqrt{1 - \frac{z}{\rho}}} + v(z) \qquad (73)$$

where $u(z)$ and $v(z)$ are regular at $z=\rho$, and $u(\rho) \neq 0$.

Since $\tau\text{reduce}(z)$ is an odd function, it has a similar expansion at $z = -\rho$, namely :

$$\tau\text{reduce}(z) = u(-z)\frac{1}{\sqrt{1 + \frac{z}{\rho}}} + v(-z) \qquad (74)$$

and $u(-\rho) = -u(\rho)$.

Explicit computation shows that :

$$u(\rho) = \frac{1}{\sqrt{2}}(2\tau + \alpha - 2\rho\theta) \quad , \qquad (75)$$

where

$$\alpha = \tau_{eq}(\rho) = \frac{2\tau}{2d(1 + \sqrt{1 - \rho^2}) - 1}$$

$$\theta = \tau\text{reduce}(\rho^2) \quad ;$$

θ can be computed numerically, for given c and d, using equation (70). The proof that $u(\rho) > 0$ is obtained by bounding terms in the expression obtained from (70). Applying the Darboux theorem and using the fact that $\tau\text{reduce}(z)$ is odd we finally obtain :

$$[z^n]\ \tau\text{reduce}(z) = \begin{cases} 0 & \text{when } n \text{ is even} \\ u(\rho)\left\{[z^n]\ (1 - \frac{z}{\rho})^{-\frac{1}{2}} - [z^n]\ (1 + \frac{z}{\rho})^{-\frac{1}{2}}\right\}(1 + O(\frac{1}{n})) & \text{when } n \text{ is odd} \end{cases}$$

Hence

$$[z^n]\ \tau\text{reduce}(z) = \begin{cases} 0 & \text{when } n \text{ is even} \\ \frac{2u(\rho)}{\sqrt{\pi}}\ \rho^{-n}\ n^{-\frac{1}{2}}\ (1 + O(\frac{1}{n})) & \text{when } n \text{ is odd} \end{cases}$$

Similarly :

$$[z^n]\, a(z) = \begin{cases} 0 & \text{when } n \text{ is even} \\ \dfrac{\tau\sqrt{2}}{\sqrt{\pi}}\, \rho^{-n}\, n^{-\frac{3}{2}}\, (1 + O(\frac{1}{n})) & \text{when } n \text{ is odd} \end{cases}$$

We therefore conclude the analysis.

Theorem 4 :

The average running time of the simplification algorithm is linear in the size of the input :

$$\overline{\tau \text{reduce}}_{2n+1} = \gamma n + O(1)$$

where the constant γ depends only on the family of trees under consideration.

With the above notations we have :

$$\gamma = 2\sqrt{2}\, \frac{u(\rho)}{\tau} \quad .$$

BIBLIOGRAPHY

[BR 82] J. BERSTEL, C. REUTENAUER : "Recognizable formal power series on trees",
Theor. Comp. Sc. 18 (1982) pp. 188-145

[Bu 85] W. BURGE : Recursive programming techniques
Addison Wesley, Reading (1975)

[Da 1878] G. DARBOUX : "Mémoire sur l'approximation des fonctions de très grands nombres, et sur une classe étendue de développements en série",
J. de Mathématiques pures et appliquées, 3ème série, tome VI, (Jan 1878) pp. 1-56

[FO 82] P. FLAJOLET, A. ODLYZKO : "The average height of binary trees and other simple trees",
JCSS, vol. 25,No 2, (Oct. 1982) pp. 171-213

[FS 70] D. FOATA, M.P. SCHUTZENBERGER : Théorie géométrique des polynômes eulériens,
Lecture Notes in Mathematics 138, Springer Verlag, Berlin (1970)

[Go 60] I.J. GOOD : "Generalizations to several variables of Lagrange's expansion, with applications to stochastic processes",
Proc. Cambridge Phil. Soc. 56 (1960) pp. 367-380

[Go 65] I.J. GOOD : "The generalization of Lagrange's expansion and the enumeration of trees",
Proc. Cambridge Philo. Soc. 61 (1965) pp. 499-517

[He 74] P. HENRICI : "Applied and computational complex analysis,
2 vol J. Wiley, New York (1974)

[JG 81] D. JACKSON, I. GOULDEN : Combinatorial Decompositions and Algebraic Enumerations (to appear)

[Kn 68] D.E. KNUTH : The art of computer programming : Fundamental Algorithms,
Addison Wesley, Reading (1968)

[Kn 73] D.E. KNUTH : The art of computer programming : Sorting and Searching,
Addison Wesley, Reading (1973)

[MM 78] A. MEIR, J.W. MOON : "On the altitude of nodes in random trees",
Canad. J. of Math 30 (1978), pp. 997-1015

[Od 82] A. ODLYZKO : "Periodic oscillations of coefficients of power series that satisfy functional equations"
Adv. in Math. 44 (1982), pp. 180-205.

[Po 37] G. POLYA : "Kombinatorische Anzahlbestimmungen für Graphen, Grupen und Chemische Verbindungen",
Acta Mathematica 68 (1937), pp. 145-254

[Ra 79] L. RAMSHAW : "Formalizing the analysis of algorithms",
Ph. D Thesis, Stanford Univ. (1979)

[Ro 75] G.C. ROTA : Finite Operator Calculs, Academic Press, New-York (1975)

[SF 82] J.M. STEYAERT, P. FLAJOLET : "Patterns and Pattern-matching in trees : an analysis", (to appear in Inf. & Control).

[Th 73] J.W. THATCHER : "Tree automata : an informal survey", in Currents in the Theory of Computing
(A.V. Aho, ed), Prentice Hall (1973) pp. 143-178.

[Vu 80] J. VUILLEMIN " A unifying look at data structures",
CACM 23 (1980), pp. 229-239.

[We 75] B. WEGBREIT : "Mechanical program analysis",
CACM 18 (1975), pp. 528-532.

[We 76] B. WEGBREIT: "Verifying program performance,
JACM 23 (1976), pp. 691-699.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique

ISSN 0249 - 6399